**MASTER EN SCIENCES INFORMATIQUES**

# Ingénierie du chaos

## Approche par rétro-ingénierie

Coppens, Patrice

*Award date:*
2019

*Awarding institution:*
Universite de Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

# Ingénierie du chaos, Approche par rétro-ingénierie

Patrice Coppens

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

## Remerciements

Travail en apparence individuel, ce mémoire est le résultat d'échanges et de recherches. Il n'aurait probablement pas pu être réalisé sans le soutient de plusieurs personnes.

Je voudrais tout d'abord adresser ma reconnaissance au promoteur de ce mémoire, Monsieur Anthony Cleve, professeur à l'université de Namur, pour sa disponibilité, sa cordialité et surtout ses remarques et questions judicieuses qui m'ont permis d'approfondir mon raisonnement et d'exprimer au mieux ma réflexion.

Je souhaite ensuite remercier le corps professoral et administratif pour la qualité des enseignements dispensés, ainsi que pour leur convivialité.

Un grand merci à Cédric, Fatiha, Anne, Lucy, Nicole et Justine pour leur travail de relecture et de correction de ce mémoire.

Je remercie également mes filles, Joey, Aaliyah, Naëlle, pour leur patience, leur compréhension et leur sacrifice.

Enfin, je ne saurais remercier assez Yasmine, mon épouse, pour son soutient et pour tout ce qui ne peut s'écrire dans un mémoire.

**Résumé**

L'ingénierie du chaos a comme objectif d'augmenter la robustesse et la confiance envers un produit logiciel, basé sur une architecture distribuée à grande échelle.
La discipline s'appuie sur la définition d'un état stable et l'assertion qu'il le restera lors de l'injection d'une faute, dans l'environnement de production.
Mais le choix de l'injection dépend de l'intuition de l'ingénieur. Est-il possible de réduire, voire d'éliminer cette approche intuitive par rétro-ingénierie du système ?

**Abstract**

Chaos engineering leads to more resilient systems and builds confidence in a software product based on a large-scale distributed architecture.
The discipline is based on the definition of a steady-state and the assertion that it will remain stable when a fault is injected in production.
But the choice of the injection depends on the engineer's intuition. Is it possible to reduce, or eliminate this intuitive approach by reverse engineering the system?

# Table des matières

# Chapitre 1

# Introduction

Aujourd'hui, un grand nombre d'applications sont basées sur une architecture distribuée. Les applications Web sont partout dans notre quotidien, sur les ordinateurs de bureau, sur les tablettes, sur les smartphones, autant dans les entreprises et les hôpitaux que dans nos foyers et nos voitures. Interconnectés, les objets sont devenus « intelligents ». Le tracteur de l'agriculteur transmet en temps réel la quantité exacte d'engrais dispersé, l'usine chimique est pilotée à distance depuis un autre continent, les jouets pour enfants et d'autres pour adultes offrent des interactions nouvelles. Notre vie dans cette partie du monde industrialisé est fortement dépendante de la disponibilité d'une multitude d'applications. Si le citoyen accepte de perdre du temps, chaque jour dans les embouteillages, il est beaucoup moins clément quand il rencontre une défaillance très temporaire dans une application d'achat en ligne ou de divertissement.
Dans l'industrie du développement informatique, le *time to market* et l'adaptabilité sont des qualités majeures pour un logiciel. Les applications monolithiques perdent leur intérêt au profit de celles fragmentées en microservices beaucoup plus petits, plus rapides et plus faciles à concevoir et à modifier.
Le style d'architecture microservices produit des applications largement distribuées tandis que celles-ci subissent des modifications plusieurs centaines de fois par jour. Les ingénieurs travaillant dans de tels systèmes les qualifient de chaotiques. En effet, tel un battement d'ailes d'un papillon au Brésil, une mauvaise configuration peut provoquer une tornade dans le système et l'indisponibilité de fonctionnalités majeures.
La très haute disponibilité, y compris pour un système largement distribué est capitale. Pour tenter d'y parvenir, une nouvelle discipline est née chez Netflix : l'ingénierie du chaos.

## 1.1 Ingénierie du chaos

L'ingénierie du chaos est basée sur deux prémisses : la première est que les ingénieurs doivent considérer l'ensemble des différents services en production, comme un seul système ; la seconde est que nous obtenons une meilleure compréhension de ce système en injectant des éléments réels et en observant ses frontières. Dans une approche traditionnelle, chaque comportement d'un logiciel est défini par une spécification fonctionnelle. En pratique, dans un système distribué, les spécifications sont incomplètes. Le problème est exacerbé par la complexité des produits, ainsi que par l'infrastructure sous-jacente.
L'ingénieur observe le système et ses frontières en prenant des mesures durant son état stable (« *steady-state* »). La préoccupation n'est plus basée sur le respect des spécifications, mais se porte sur le bon fonctionnement général et sur les plaintes des utilisateurs.
Pour vérifier la robustesse du système, l'ingénieur va injecter une faute et vérifier qu'il se comporte normalement. L'injection de fautes, est un processus coûteux et risqué qui doit être mûrement réfléchi. Si l'injection ne provoque aucun soubresaut visible, elle sera une perte de temps et d'énergie. Si elle provoque au contraire une panne majeure, elle sera responsable d'une perte de confiance, de temps et d'énergie. L'injection de fautes a un grand intérêt si celle-ci affecte significativement le système sans provoquer la moindre panne, visible pour l'utilisateur.
Dans l'ingénierie du chaos, le choix de la cible de l'injection de fautes est, malgré les enjeux, purement intuitif.

## 1.2 Faiblesse de l'ingénierie du chaos

Une grande faiblesse de l'approche intuitive de l'ingénierie du chaos est le manque de scalabilité qu'entraîne l'utilisation d'un personnel très qualifié, connaissant en profondeur le système, tant techniquement que fonctionnellement. De plus, l'ingénieur est sujet aux biais cognitifs qui vont influencer son jugement.
Aujourd'hui, il n'existe pas de système expert, d'intelligence artificielle capables de découvrir une faiblesse dans un système largement distribué.

## 1.3 Rétro-ingénierie

La rétro-ingénierie est une discipline qui consiste à découvrir le fonctionnement interne d'un artefact. Une des méthodes possibles est basée sur l'observation des communications du système. Elle se décompose en trois étapes. La première collectera les informations en utilisant les logs produits par le système. L'étape suivante devinera les fonctionnalités du système. La

dernière étape offrira une représentation visuelle de ce dernier.

## 1.4 Ingénierie du chaos, approche par rétro-ingénierie

Ce mémoire a comme objectif de répondre aux questions suivantes :

— En utilisant une méthode de rétro-ingénierie, est-il possible de réduire voire supprimer l'approche intuitive lors de la sélection judicieuse d'une cible de l'injection de fautes ?
— Cette méthode offre-t-elle une aide à la décision qui permet de réduire les connaissances nécessaires à cette sélection ?

# Chapitre 2

# Structure du document

Ce document est organisé en trois parties.

## 2.1 Ingénierie du chaos, état de l'art

La première partie présente l'état de l'art de l'ingénierie du chaos. Elle est constituée des éléments suivants :

- Une introduction à la notion de chaos.
- L'état de l'art de l'ingénierie du chaos : La présentation en détail de cette discipline, les problèmes qu'elle adresse, son champ d'application et ses limitations.
- Pour une méthode de rétro-ingénierie, les notions théoriques et les connaissances actuelles pertinentes pour la suite du document

Ces éléments vont servir de fondations à la partie suivante.

## 2.2 Ingénierie du chaos, approche par rétro-ingénierie

La deuxième partie présente une approche par rétro-ingénierie. Elle est composée des chapitres suivants :

- Une présentation générale.
- Une présentation détaillée de la méthode et les différentes étapes du processus.
- L'application de la méthode dans deux études de cas.

## 2.3 Discussion, perspectives

Cette dernière partie analyse les éléments précédents et présente :

- Les limites de l'approche.
- Les perspectives de la méthode de rétro-ingénierie sélectionnée.
- Les conclusions du mémoire.

# Première partie

# Ingénierie du chaos, état de l'art

**Cette partie présente l'état de l'art de l'ingénierie du chaos et les éléments existants utilisés en support pour la suite du mémoire. Elle est divisée en quatre chapitres. Le premier introduit ce mémoire. Le deuxième décrit les problèmes que cette discipline adresse et son champ d'application. Le troisième présente l'état de l'art de l'ingénierie du chaos. Le dernier propose des techniques de rétro-ingénierie.**

# Chapitre 3

# Problèmes adressés

## 3.1 Concepts utilisés

Ce chapitre présente les concepts communément rencontrés lorsque l'on aborde l'ingénierie du chaos ou la rétro ingénierie. Ils ne sont pas propres à cette discipline, mais peuvent parfois avoir une signification particulière.

### Conteneur

Un conteneur logiciel est un environnement d'hébergement pour composants logiciels. Il fournit un support d'exécution similaire à un système d'exploitation, ainsi que de la surveillance et des contraintes sur les interactions entre les composants hébergés et leurs clients. SridharJason and Hallstrom (2006)

### 3.1.1 Corrélation ID

L'ID de corrélation est un patron d'intégration (*Enterprise Integration Patterns*). « [...] un message peut avoir [...], un identifiant unique comme la clé primaire pour une ligne dans une table de base de données relationnelle. Cet identifiant unique pourrait être utilisé pour identifier le message d'autres messages. [..]
Voici comment fonctionne un identifiant de corrélation : lorsque le demandeur crée un message de requête, il attribue à la demande un identifiant de demande - un identifiant différent de ceux de toutes les autres demandes en attente [...]. Lorsque le répondant traite la requête, il enregistre l'ID de la demande et l'ajoute à la réponse en tant qu'ID de corrélation. Quand le demandeur traite la réponse, il utilise l'ID de corrélation pour savoir à quelle demande la réponse s'adresse. Ceci est appelé un identifiant de corrélation en raison de la façon dont l'appelant utilise l'identifiant pour corréler (par exemple, faire correspondre ; afficher la relation) chaque réponse à la demande qui l'a provoquée. [...]

Un identifiant de corrélation [...] est généralement placé dans l'en-tête d'un message plutôt que dans le corps. L'ID ne fait pas partie de la commande ou des données que le demandeur tente de communiquer au répondant. En fait, le répondant n'utilise pas du tout l'identifiant ; il enregistre simplement l'ID de la demande et l'ajoute à la réponse dans l'intérêt du demandeur. Puisque le corps du message est le contenu en cours transmis entre les deux systèmes, et que l'ID ne fait pas partie de celle-ci, l'ID va dans l'en-tête. » Hohpe and Woolf (2003)

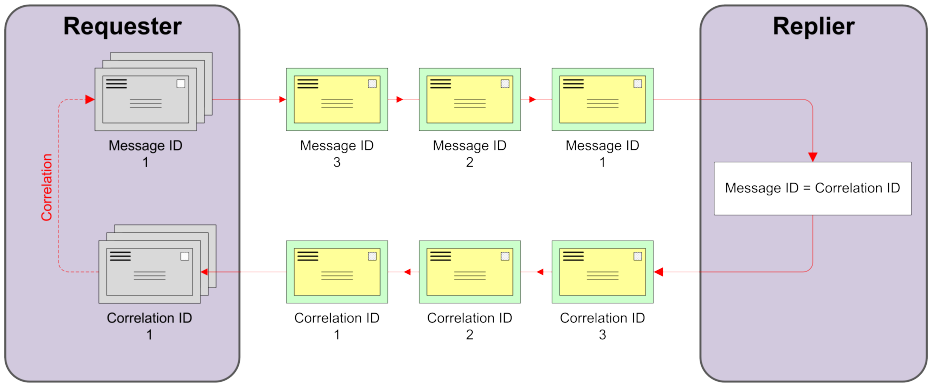


FIGURE 3.1 – ID de corrélation UniKnow (2016)

Si certaines applications ne produisent pas d'ID de corrélation dans leurs logs, « pour permettre aux logs de plusieurs services d'être analysés ensemble, la passerelle API doit fournir un identifiant de transaction injecté dans les en-têtes de requête afin que les services en aval puissent l'inclure lors de la journalisation de leurs activités. » Gámez-Díaz et al. (2015)

### *Devops*

Le *devops* est une culture qui vise, dans un système d'information, à réunir dans une même équipe des compétences de développement et des compétences opérationnelles. Historiquement, les préoccupations concernant la construction d'un produit logiciel et les préoccupations concernant le maintien du produit dans l'environnement de production sont distinctes. Pour répondre aux demandes de changement de plus en plus rapide, un rapprochement est nécessaire.
Le *devops* est aussi un état d'esprit de travail d'équipe sur un objectif commun, qui tente d'effacer les préoccupations de service, de direction ou de département. On y inclut parfois la gestion financière et le marketing. L'organisation en *devops* est jugée indispensable pour la réussite de l'adoption de l'architecture microservices. Mazzara and Meyer (2017)

### 3.1.2 Journalisation des données

Selon l'article Sendmail (2007) « Importance du serveur de journalisation centralisé et des logiciels d'analyse de journaux pour une organisation »R.Anusooya1 et al. (2015), « les journaux des serveurs sont la source d'information la plus riche dans l'organisation de l'entreprise » (traduction libre) Les logs (journaux) applicatifs sont produits et enregistrés dans un système de persistance, habituellement un système de fichiers ou une base de données. L'article de R.Anusooya1 et al. (2015) présente en détail la récolte, la rétention, la centralisation et l'exploitation des logs dans l'entreprise. Le processus peut se résumer comme suit :

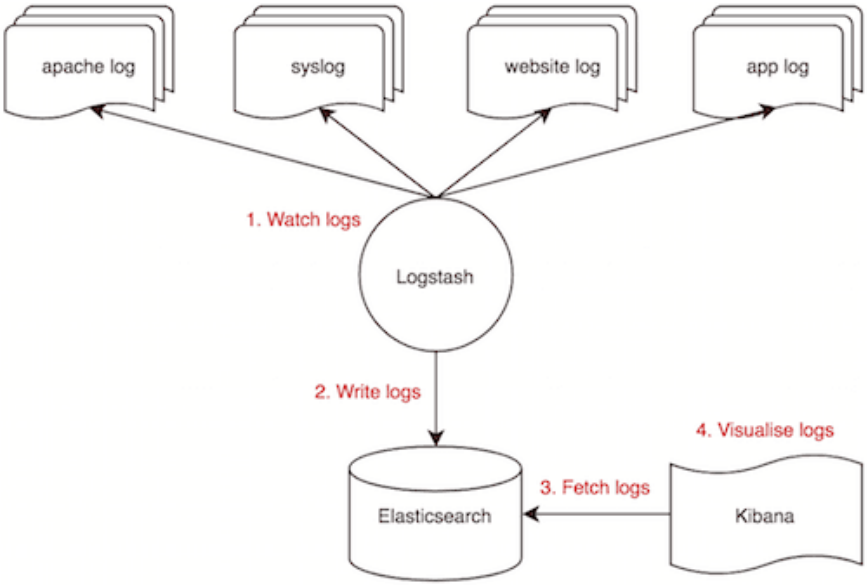


FIGURE 3.2 – Flux de la journalisation Inanzzz (2017)

Les applications produisent des logs qui sont ensuite centralisés, organisés, filtrés et enregistrés dans un système de recherche, par exemple une base de données NO-SQL. Une application de requête et une interface de visualisation permettent d'exploiter les logs.
Exemple de logs :

Listing 3.1 – nginx log

```
172.17.0.1 - - [29/Oct/2018:11:23:03 +0000] "GET /
    number.html HTTP/1.1" 200 1 "-" "Mozilla/5.0 () "
172.17.0.1 - - [29/Oct/2018:11:23:03 +0000] "GET /
    favicon.ico HTTP/1.1" 404 555 "http
    ://192.168.1.15:8081/number.html" "Mozilla/5.0 () "
```

### Livraison continue (CI/CD)

La livraison continue (*Continuous Delivery*, CD) est une pratique qui vise à livrer fréquemment un produit logiciel de qualité. Pathania (2017) L'intégration continue (CI) est une pratique qui, lors de la réalisation du code source, exécute les tests unitaires et les tests d'intégration afin d'assurer la qualité et la bonne intégration de ce code source. Rahman et al. (2017) Les deux pratiques utilisent des outils d'automatisation tels que *Jenkins* Kawaguchi (2011), *Visual Studio Team Services* Microsoft (2012) et *Spinnaker* Netflix (2015).

### Microservices

Les microservices sont un paradigme de développement émergent dans lequel un logiciel est obtenu par la composition d'entités autonomes, appelées microservices. Cependant, les systèmes basés sur les microservices sont actuellement développés en utilisant les langages de programmation généralistes qui ne fournissent aucune abstraction pour la composition des services. La pratique actuelle vise plutôt à se focaliser sur l'aspect de déploiement, en particulier par l'utilisation de conteneurisation. Mazzara and Meyer (2017)

### Netflix

Netflix, Inc. (« constitué en société commerciale ») est un réseau mondial de télévision par Internet, avec plus de 93 millions de membres à travers 190 pays distribuant plus de 125 millions d'heures de film et de divertissement par jour. Les membres peuvent visionner le contenu sur pratiquement n'importe quel écran connecté à Internet. Netflix (2016)

### Portail (*Gateway, API Manager*)

Un portail (*API Gateway*) est le point d'accès unique pour accéder aux microservices. Il offre des fonctionnalités de gestion telles que l'authentification des utilisateurs, la gestion des versions des API, leur mise en cache ainsi que le support des plans de tarification. Gámez-Díaz et al. (2015)

### Produit logiciel

Un produit logiciel est un concept marketing. Il désigne quelque chose qui peut être offert sur le marché pour un usage qui satisfait un désir ou un besoin, contre une forme de compensation. Le produit peut être un objet physique, des idées ou un mix de ces entités. Maedche et al. (2012)

### 3.1.3 Rétro-ingénierie

« Le Reverse Engineering, ou rétro-ingénierie / ingénierie inverse en français, représente l'étude et l'analyse d'un système pour en déduire son fonctionnement interne, et se retrouve dans de nombreux domaines de l'ingénierie : génie civil, mécanique, ingénierie navale, aéronautique, etc.[...] Dans le domaine de l'informatique, le reverse a de nombreuses utilités : [...] Il peut être utilisé pour comprendre en détail le fonctionnement d'un logiciel, » (ESIPE, 2013)

#### Scalabilité

Selon la taille du système largement distribué, la gestion des logs peu représenter un véritable défi. Selon Lew and Narayanan (2017), « À un moment donné de la croissance de notre entreprise, nous avons appris que le stockage des logs d'application bruts ne serait pas scalable. À mesure que les applications migraient vers une architecture de microservices, nous avions besoin d'un moyen pour mieux comprendre les décisions complexes prises par les microservices. Le suivi des demandes distribuées est un début, mais n'est pas suffisant pour bien comprendre le comportement des applications et la résolution des problèmes. L'augmentation de la trace de la demande avec le contexte d'application et des conclusions intelligentes est également nécessaire. »

### 3.1.4 Style d'architecture REST

« Un service Web RESTful est un service Web simple implémenté à l'aide de HTTP. [. . . ] REST est plus un ensemble de principes qu'un ensemble de normes. Hormis ses six contraintes globales, rien n'est dicté. Il existe des « meilleures pratiques » et des normes de facto, mais celles-ci évoluent constamment [. . . ].
La caractéristique principale de l'architecture REST est qu'elle adopte une « vue ressource » du monde.
Les principes RESTful sont :

- P1 : La ressource peut être identifiée par un URI (*Uniform Resource Identifier*, identifiant uniforme de ressource) ;
- P2 : Séparation de la ressource abstraite et de ses représentations concrètes ;
- P3 : Interaction sans état, chaque interaction contient toutes les informations de contexte et métadonnées nécessaires ;
- P4 : Petit nombre d'opérations, avec une sémantique distincte basée sur les méthodes HTTP : opérations sécurisées (*Get, Head, Options, Trace*) ; opérations non sécuritaires et idempotentes (*Put, Delete*) ; et opération non sécurisée et non idempotente (*Post*) ;

— P5 : Support de la mise en cache pour les opérations idempotentes et les métadonnées de représentation ;
— P6 : Promouvoir la présence d'intermédiaires tels que les mandataires, les passerelles ou les filtres pour modifier ou restreindre les demandes et les réponses en fonction de métadonnées. » Guinard et al. (2016)

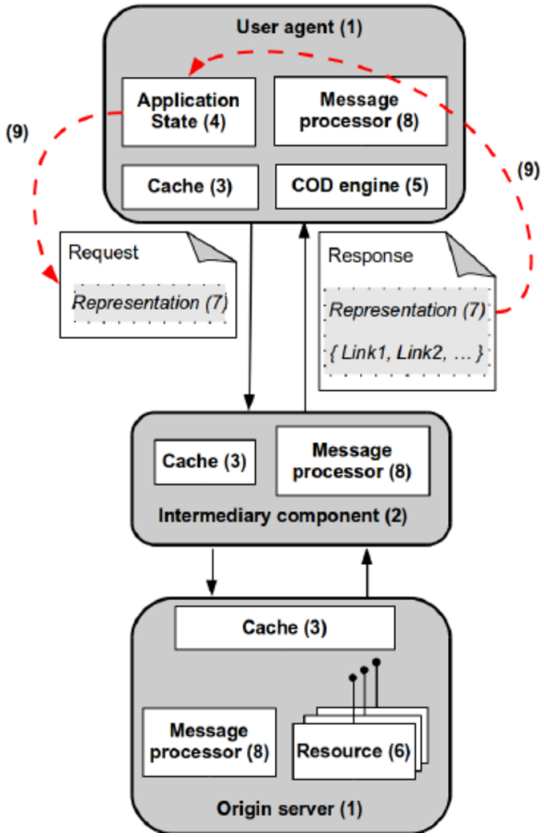


FIGURE 3.3 – Visualisation simplifiée des principes REST Guinard et al. (2016)

## Tests de logiciel

Les tests jouent un rôle essentiel dans la vérification de la correction d'un logiciel et dans la confirmation que les spécifications ont correctement été implémentées. Les tests logiciels confirment qu'un produit est prêt à être rendu public. G. (2017)

### 3.1.5 URI

Dans le style d'architecture REST, chaque ressource est identifiée par un URI. Un URI doit respecter la norme RFC 3986 qui peut se résumer dans la formule suivante :

```
scheme://host[:port][/path]["?"query]
```

- « scheme » représente le protocole (http(s), file, ...),
- « host[ :port] » est l'autorité, élément qui va gérer la suite de l'URI (généralement une adresse d'un serveur ou son nom),
- « path » contient des données qui identifient la ressource,
- « query » selon la RFC, « sert à identifier une ressource dans le cadre du schéma et de l'autorité de nommage de l'URI »

### 3.1.6 Théorème de Bayes

Dans son livre « La Formule Du Savoir, une philosophie unifiée du savoir fondée sur le théorème de Bayes » Lê Nguyên Hoang présente le théorème de Bayes, un théorème mathématique découvert par Thomas Bayes et retrouvé indépendamment par Pierre-Simon Laplace. « Une alternative à la méthode scientifique [...] la théorie statistique de la décision montre que les inférences bayésiennes sont essentiellement les seules méthodes d'apprentissage admissibles, dans le sens où une méthode n'est jamais dominée par une autre, si et seulement si, elle revient à appliquer la formule de Bayes » Hoang (2018)

La formule de Bayes : $P(T|D) = \frac{P(D|T)P(T)}{P(D)}$

Soit, $P(X|Y)$ pour probabilité de $X$ en connaissant $Y$, appelé également loi a posteriori :
la probabilité de la théorie T en connaissant les données D est égale à :
la probabilité d'avoir les données $D$ en sachant la théorie $T$ (la vraisemblance),
multipliée par la probabilité d'obtenir la théorie $T$ (loi a priori),
divisée par la probabilité d'obtenir $D$.

Exemple tiré de *La Formule du Savoir* :
« Rentrant du Nigéria, un diagnostic affirme que vous êtes atteint d'Ebola. Lorsqu'une personne est saine, le taux de diagnostics corrects est de 90%. Devez-vous commencer à écrire votre testament ?
En Afrique Noire, pas plus d'une personne sur 10.000 est victime d'Ebola.
Notons : $E$ être porteur d'Ebola, $S$ être sain, $d$ un diagnostic défavorable et $f$ un favorable :

$$P(E|d) = \frac{P(d|E)P(E)}{P(d|E)P(E) + P(d|S)P(S)}$$
$$P(E|d) = \frac{1 \cdot 0.0001}{1 \cdot 0.0001 + 0.1 \cdot 0.9999}$$
$$P(E|d) \approx 0.001$$

La probabilité d'avoir contracté la maladie est, selon le diagnostic, de 1 pour mille. »(Ibid.)

### Approximation de Bayes

Dans « La Formule Du Savoir, une philosophie unifiée du savoir fondée sur le théorème de Bayes » Lê Nguyên Hoang (ibid.) propose cinq approches pour passer d'un bayésianisme pur à un bayésianisme pragmatique.
«Une première approche consiste à considérer uniquement un nombre restreint de modèles candidats. [...] Cette approche pourrait être typiquement complétée par l'algorithme par *multiplicative weights update*. [...]
Une deuxième approche consiste à ne calculer qu'un modèle à forte crédence, voire à identifier le modèle le plus crédible. C'est ce que l'on appelle la maximisation de l'a posteriori (MAP). [...]
Une troisième approche consiste à ignorer la fonction de partition, ce dénominateur de la formule de Bayes qui requiert la comparaison de tous les modèles imaginables. [...]
La quatrième approche que j'aimerais mentionner est la plus étrange. Elle consiste à s'autoriser de violer les lois des probabilités. [...] Cette largesse semble conduire à des résultats qui ont l'avantage d'être rapide à calculer. [...]
Enfin une cinquième et dernière approche consiste à considérer un ensemble restreint de lois de probabilité. De façon cruciale, il est alors important de disposer d'une mesure de similarité entre des lois de probabilité. »

### Modèle à forte croyance

Pour calculer un modèle à forte croyance (*credence*, en anglais), plusieurs algorithmes sont utilisés, notamment les réseaux antagonistes génératifs (*generative adversarial networks* (GANs), en anglais) qui ont obtenu d'énormes succès dans les domaines de la vision par ordinateur, de la classification des images, du traitement de la parole et du langage naturel. Ceux-ci demandent un large jeu de données et une puissance de calcul conséquente, ce qui dépasse le cadre d'une rétro-ingénierie.

Les modèles d'inférence pour identifier deux services équivalents ne sont heureusement pas aussi complexes que ceux utilisés dans la classification d'images. Un microservice est une unité exécutable de déploiement qui, lors de son utilisation dans un système, va être instancié plusieurs fois dans des conteneurs différents. Les URI des endpoints sont alors identiques, à l'exception de leur autorité. Selon les pratiques de l'organisation, il peut être en revanche hasardeux d'utiliser le *path* pour inférer sur une quelconque équivalence. Un *path* de cinquante caractères contenant « v1 » pour indiquer la version première du service à une distance de Levenshtein normalisée de 0,02 (0 indiquant totalement identique et 1 totalement différent) par rapport au *path* qui change la version en « v4 ». Cette faible distance ne reflète pas la grande différence de fonctionnalité et toutes les incompatibilités entre les deux services. À contrario, «/cust » et «/distribuedMemoryCache-cust » peuvent être des *paths* de services équivalents alors qu'ils ont une distance de Levenshtein normalisée de 0,85.

Après cette liste de concepts permettant de mieux aborder l'ingénierie du chaos et les principes de rétro-ingénierie, la section suivante va tenter de répondre à la question : « qu'est-ce que le chaos ? »

## 3.2 Notion de Chaos

« Durant le 20ème siècle, trois grandes révolutions se sont produites : la mécanique quantique, la relativité et le chaos. La théorie du chaos, aussi appelée théorie des systèmes dynamiques, est l'étude de comportement apériodique instable dans un système dynamique déterministe, qui montre une dépendance sensible aux conditions initiales (Vaidyanathan, 2013).
« La dépendance sensible aux conditions initiales implique que des conditions initiales arbitraires suivent des trajectoires qui divergent après un certain temps » (Moon, 2008) [...]. En raison du déterminisme (Morrison, 2012), le chaos est prévisible à court terme, mais il est imprévisible à long terme en raison de la sensibilité aux conditions initiales. Le chaos se caractérise par une forte dépendance sensible à l'état initial, par son incapacité à prédire les conséquences futures, par l'exposant de Lyapunov (Kuznetsov, 2016), par sa dimension fractale, etc.
La dynamique non linéaire et les termes de chaos ont été compris de la plupart des scientifiques et des ingénieurs au cours des dernières décennies. Les non-linéarités se produisent dans les processus de rétroaction (Gaponov-Grekhov et Rabinovich, 2011), dans les systèmes contenant des sous-systèmes en interaction et dans les systèmes en interaction avec l'environnement.
Ce scénario est qualitativement et quantitativement distinct des situations où les perturbations se développent de manière linéaire. Grâce à la disponibilité des ordinateurs à grande vitesse et aux nouvelles techniques d'analyse, il est devenu évident que le phénomène chaotique est de nature universelle et a des conséquences transversales dans divers domaines de l'activité humaine. [...]
Une propriété clé du chaos est que de simples systèmes dynamiques peuvent souvent engendrer des dynamiques complexes. [...]
Les mouvements de plusieurs systèmes naturels ou techniques peuvent être régis par un ensemble d'équations dérivées de lois naturelles telles que les lois de Newton ou l'équation d'Euler. Les équations qui décrivent un système dynamique peuvent être algébriques ou différentielles. L'ensemble des équations, définies mathématiquement comme un système dynamique, donne l'évolution temporelle de l'état d'un système à partir de la connaissance de son histoire antérieure. Par conséquent, l'état à tout moment peut être déterminé par les équations de gouvernance et les états initiaux. Dans la science moderne, le terme chaos est utilisé pour décrire un type de mouvement résultant d'un système dynamique qui semble désordonné et extrêmement complexe à l'examen approfondi. La complication et le désordre sont dus au fait que le chaos est un mouvement apériodique récurrent. Par conséquent, le chaos peut être défini comme une réponse d'état stationnaire liée qui n'est pas un état d'équilibre, un mouvement périodique ou un mouvement quasi périodique.
Les mouvements chaotiques sont également caractérisés par une sensibilité

aux états initiaux ; c'est-à-dire qu'une simple variation des conditions initiales peut rapidement produire d'énormes différences de réponse. La prédiction à long terme du chaos est impossible, en raison d'une telle sensibilité. En d'autres termes, le chaos est imprévisible après un certain temps, car une petite différence dans les conditions initiales, au-delà de leur précision, entraînera une croissance rapide du mouvement.
Un système dynamique est qualifié de chaotique s'il satisfait les trois propriétés : limitation, récurrence infinie et dépendance sensible aux conditions initiales (Azar et Vaidyanathan, 2015, Azar et Vaidyanathan, 2014). La théorie du chaos concerne l'étude qualitative et numérique du comportement apériodique instable dans des systèmes dynamiques non linéaires déterministes.» (Traduction libre, Nasr and Mekki (2018))

Dans sa thèse de doctorat, Imen Ellouse décrit :
« La notion de stabilité constitue une problématique centrale de la théorie du contrôle. Souvent liée à la façon d'appréhender un système, la stabilité possède un large éventail de définitions [...].
Il est frappant de noter que des dispositifs simples, tels que le double pendule, et un événement très complexe, tel que la météo, suivent la même dynamique, qui ne peut être prédite que pour de courtes périodes (Kaygisiz et al., 2011). [. . . ] » (Traduction libre, Nasr and Mekki (2018))

Quand un système est qualifié de chaotique, comme le sont certains systèmes largement distribués, quels types de problème spécifiques apportent-ils ? La section suivante va présenter ceux que l'ingénierie du chaos tente de résoudre.

## 3.3 Le problème adressé

### Résilience et haute disponibilité

Agilité et évolutivité, sont deux des qualités qui sont mises en avant pour expliquer la transformation d'un système monolithique en un système largement distribué, utilisant l'architecture microservices.
Le monde financier est soumis à l'imprévisibilité des marchés Parrish and Halsey (2015), tandis que les entreprises basées sur la technologie internet voient leur croissance exploser. En cinq ans, le nombre d'abonnements Netflix est passé de 26,5 millions à 117 millions. Les termes mêmes de croissance et d'évolutivité portent de nouvelles significations.
La scalabilité verticale, c'est-à-dire l'augmentation des capacités physiques d'un serveur, est par définition limitée. Le coût, bien avant les limitations techniques, d'un tel mode d'évolutivité est bien souvent un frein à sa sélec-

tion.
Pour un large éventail d'activités -la vente, le divertissement, et les services-, la capacité à fournir rapidement un nouveau contenu prend une importance capitale. Les applications dites monolithiques sont peu adaptées à des changements fréquents. Ils sont planifiés avec soins selon un processus strictement défini, dont la temporalité entre les livraisons des modifications se compte en mois. Pour répondre à ces besoins d'agilité et d'évolutivité, une application est distribuée en un nombre conséquent de microservices. Plusieurs centaines de ceux-ci pour des milliers d'instances de machines virtuelles. Evans (2016) Blohowiak et al. (2016)
Même si tous les systèmes d'information n'atteignent pas ce gigantisme, la multiplication des services et de leurs problèmes potentiels tend vers un système plus fragile. En effet, 30 dépendances avec une disponibilité de 99,99% donnent une disponibilité totale de 99,7% ($99.99^{30}$). Une des réponses à ce problème est d'ajouter des mécanismes de redondance pour augmenter la résilience et garantir une haute disponibilité.

### Nature chaotique

Dans son article « *No Silver Bullet : Essence and Accidents of Software Engineering* » (traduction libre : *Pas de balle d'argent : essence et accidents de l'ingénierie logicielle*), Brooks définit la complexité d'un logiciel comme suit : « Une entité logicielle est plus complexe que peut-être toutes autres constructions humaines car il n'y a pas deux composants qui soient identiques. [...] La complexité d'un logiciel n'est pas accidentelle, c'est une propriété essentielle. Une description d'un logiciel qui fait abstraction de sa complexité fait souvent abstraction de son essence. »
De la complexité découle la difficulté d'énumérer et de comprendre les états possibles de celui-ci. De la complexité vient la difficulté d'étendre le système à de nouvelles fonctionnalités, sans créer d'effets de bord. Brooks (1987)
Pour Foote Brian et Yoder Joseph, les systèmes distribués sont probabilistes par nature et sont sans doute mieux modélisés de façon probabiliste. Foote and Yoder (2003)

### Gestion de la complexité

Gérer la complexité est un défi et une opportunité pour les ingénieurs. Les opportunités pour innover et optimiser un système complexe sont immenses. Habituellement, les ingénieurs logiciels optimisent trois propriétés : la performance, la disponibilité et la résilience.
Dans ce contexte, la performance se réfère à la minimisation de la latence et de son coût. La disponibilité désigne la capacité à éviter les arrêts, planifiés et non planifiés. La résilience concerne la capacité à retrouver un état stable suite à un événement indésirable.

Lors de l'adoption de l'architecture microservices, une quatrième propriété est considérée : la vélocité à fournir une fonctionnalité. Elle décrit la vitesse avec laquelle l'équipe de développeurs peut fournir aux clients une fonctionnalité innovante. Dans une organisation en *devops*, les équipes sont largement indépendantes et décident quand elles vont déployer leurs changements dans l'environnement de production. Elles partagent néanmoins une vision globale et ont la capacité de contribuer à l'objectif commun. La communication y joue un rôle essentiel.
L'ingénierie du chaos supporte la vélocité en apportant une grande confiance aux équipes, à travers la vérification de la résilience.

### Comprendre les systèmes complexes

L'architecte informatique a dans ses responsabilités la compréhension du système. Mais quand la taille de celui-ci augmente, *a fortiori* quand il est largement distribué, il devient rapidement trop complexe pour la compréhension humaine.

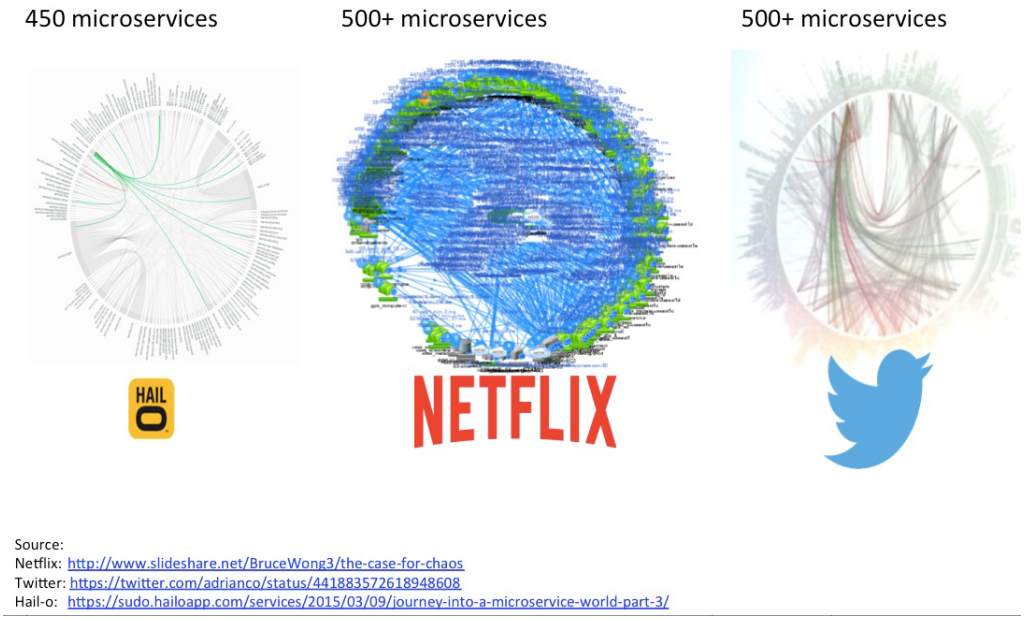


FIGURE 3.4 – Complexité d'architecture microservices

Il faut également prendre en compte que le système évolue sans cesse, parfois plusieurs fois par jour.
L'architecture microservices apporte un gain de vélocité et de flexibilité, au prix d'une perte de la compréhension du système. Mais les systèmes

monolithiques ne sont pas épargnés par l'augmentation de la complexité. De même certains systèmes sont, par conception, extrêmement complexes : réseaux neuronaux, algorithmes d'intelligences artificielles, etc.
L'ingénierie du chaos permet d'apprivoiser ces systèmes et de mettre en avant ces propriétés.

#### Tests exhaustifs

Si un composant unique peut être testé, voir vérifié formellement, peut-on envisager de tester toutes les combinaisons possibles d'un système complexe ? Selon le *Linux journal* Strauss (1998), pour faire revivre le Titanic en 1997, 200 serveurs alpha DEC ont formé un système distribué pour générer les images de synthèse du paquebot qui allaient servir dans le film. Peut-on utiliser la formidable puissance disponible dans le *Cloud* pour effectuer des tests exhaustifs ?
Dans l'article « *Test them all* » Halin et al. (2017) (traduction libre : *Testez-les tous*), les auteurs ont testé toutes les combinaisons de l'outil JHipsterJHipster (2013). Celui-ci est un générateur de squelettes d'application basé sur le langage Java et le *framework* Spring-Boot.
Les auteurs ont testé plus de 26 000 combinaisons différentes, ce qui leur a permis de découvrir que 35,70% des combinaisons comportaient des erreurs. Mais, malgré une optimisation des tests, ceux-ci ont dépassé le budget prévu. Il aurait fallu 8 mois-hommes et environ l'équivalent de 4.376 heures (182 jours) d'occupation CPU, soit plus de 54 heures sur 80 machines.
Même si l'environnement de test avait été sans frais, ce qui était loin d'être le cas, les développeurs de JHipster ont estimé que les délais induits par les tests ne seraient pas envisageables.

Cette section vient de lister les défis propres aux systèmes largement distribués que tentent d'adresser l'ingénierie du chaos. La section suivante présente ce qui peut être considéré comme l'ancêtre de cette discipline : l'injection de fautes.

## 3.4 Injection de fautes (*Fault injection*)

### Principes de l'injection de fautes

Selon Barbosa Raul, Karlsson Johan, Madeira Henrique et Vieira Marco, la résilience des systèmes est conçue pour que ceux-ci soient opérationnels malgré la présence d'un niveau acceptable d'événements fautifs ou adverses. Évaluer la résilience d'un système donné demande que les effets de tels événements puissent être examinés et mesurés en détails, ce qui nécessite la capacité à introduire des défauts et à observer leurs conséquences sur le fonctionnement du système. Barbosa et al. (2012)

L'injection de fautes est la méthode fondamentale pour l'évaluation de la résilience. Elle permet l'étude des effets de fautes sur le système, l'identification des faiblesses de la gestion de celles-ci, l'évaluation effective de la détection et des mécanismes de tolérance aux fautes, ainsi que la quantification des effets de ces fautes sur la capacité du système à fournir le service attendu.
Selon Voas Jeffrey Voas (1998), les premières injections de fautes datent de 1970, faisant probablement de cette technique l'ancêtre de l'ingénierie du chaos.

### Différences avec l'ingénierie du chaos

L'ingénierie du chaos et l'injection de fautes partagent largement les mêmes préoccupations et le même outillage. L'ingénierie du chaos est une pratique qui génère de nouvelles informations, alors que l'injection de fautes se concentre sur l'évaluation de conditions particulières. L'ingénierie du chaos couvre, par exemple, l'expérimentation d'une large et rapide augmentation du trafic, ce qui n'est, à proprement parlé, pas une condition fautive pour un site de commerce en ligne. L'injection de fautes se focalise sur des erreurs déjà rencontrées ou imaginées par les ingénieurs sur la base de cas probables ou survenus. Elle n'a pas pour objectif d'explorer des événements imprévisibles.
La principale différence des deux approches est la même que celle que l'on peut distinguer entre réaliser un test et réaliser une expérimentation. Le test se base sur le triplet de Hoare {P} S {Q}, où P représente les préconditions, S est l'exécution du système et P désigne les postconditions. Ajoutons qu'un test doit se terminer et nous obtenons ainsi une correction totale. Un test donne une information binaire : réussite ou échec. Il n'apporte pas de connaissance sur le système.
L'expérimentation quant à elle génère des informations qui posent de nouvelles interrogations et amènent à de futures expérimentations.

La section suivante présente succinctement l'ingénierie du chaos.

## 3.5 Le concept d'ingénierie du chaos

### Une discipline

Selon les auteurs du livre *Chaos Engineering, Building Confidence in System Behavior through Experiments* (traduction libre : *Ingénierie du chaos, bâtir la confiance d'un système à travers l'expérimentation*), l'ingénierie du chaos est une discipline d'expérimentation qui permet d'augmenter la confiance envers les capacités d'un système distribué, à travers les conditions turbulentes de production. Rosenthal et al. (2017)

Les systèmes distribués contiennent tellement de composants interagissant que le nombre de choses qui peut mal se passer est énorme : du disque dur qui refuse de fonctionner, en passant par un réseau en panne, un soudain pic du nombre d'utilisateurs ou des performances qui se dégradent.
S'il est impossible de prévenir toutes les pannes potentielles, les ingénieurs peuvent identifier des faiblesses et les adresser avant qu'une panne ne se produise. Ils rendent ainsi le système plus résilient et bâtissent la confiance envers celui-ci. L'ingénierie du chaos est une méthode expérimentale qui met en lumière les faiblesses systémiques de l'infrastructure.
Elle ne vise pas à causer des interruptions de service. Même s'il est aisé de casser le système, ce n'est pas toujours très productif. L'ingénierie du chaos s'intéresse davantage à faire émerger le chaos déjà présent dans un système complexe. Apporter une meilleure compréhension des effets systémiques permet de concevoir des systèmes distribués plus robustes.
L'ingénierie du chaos est une approche qui permet d'apprendre comment se comporte le système, en appliquant une discipline d'exploration empirique. À l'instar des scientifiques qui conduisent une expérience pour étudier des phénomènes sociaux, les ingénieurs du chaos mènent une expérience pour comprendre les particularités d'un système.

**Une exclusivité Netflix ?**

À travers le monde, divers événements sont organisés pour permettre le partage de pratiques dans le domaine des sciences informatiques. Celles dédiées à l'ingénierie du chaos rassemblent des pratiquants venant de Google, Amazon, Microsoft, Dropbox, Yahoo !, Uber, Gremlin Inc., O'Reilly Media, Visa, ainsi que de plusieurs universités.
Les offres d'emploi concernant cette pratique commencent à émerger. Sur des sites de recherches tels que Monster.com, la requête, effectuée le 4 avril 2018, comprenant « *chaos engineer* » renvoie à 426 offres. A titre de comparaison, sur le même site, « VSTS », un outil de gestion de développement de Microsoft, donne 845 offres. Les requêtes spécifiant un langage de programmation renvoient à plus de 1 000 offres.

# Chapitre 4

# Ingénierie du chaos

Ce chapitre présente l'état de l'art de l'ingénierie du chaos.
Il est divisé est plusieurs sections :

- Méthodologie de recherche utilisée.
- Les principes de l'ingénierie du chaos.
- Exécuter une expérimentation chaotique.
- Prérequis et outillage.
- Variantes de l'ingénierie du chaos.
- Les limites de l'ingénierie du chaos.
- Lineage-Driven Fault Injection (LDFI).
- Critiques du LDFI.

## 4.1 Méthodologie de recherche

La méthodologie enseignée pour réaliser un état de l'art consiste à effectuer une recherche préliminaire puis à préciser celle-ci pour réduire le nombre de documents pertinents. Avec une discipline naissante, cette méthodologie est confrontée à un trop petit nombre de sources. Une autre méthodologie à donc été suivie. Elle est présentée dans cette section.

### 4.1.1 Stratégie de recherche

La stratégie de recherche est en partie tirée de la *Méthodologie et protocole de recherche* de Remi Bachelet Bachelet (2015). La « démarche vise la compréhension d'un phénomène. Par là on peut l'apparenter à une étude de définition. Il s'agit de distinguer un phénomène nouveau et de rendre intelligible son fonctionnement. » (Ibid.) Dans leur contribution à *Case Research* Post et Andrew (cités par Bachelet (2015)) distinguent les catégories de recherche suivantes : décrire, explorer, expliquer ou prédire.

La stratégie de recherche combine la recherche descriptive qui « cherche

à articuler en une image cohérente la collection de faits qui se manifestent dans un objet d'étude » Bachelet (2015) et la recherche exploratoire qui « cherche à identifier des problèmes ou propriétés de situations ou événements complexes » (Ibid.).
La stratégie de recherche a pour objectif la compréhension de l'ingénierie du chaos et l'identification de ses limites.

### 4.1.2 Protocole de recherche

Une revue systématique de la littérature a été menée sur les bibliothèques et portails digitaux scientifiques. Les résultats retenus devaient concerner l'exécution de tests de produit logiciel, directement sur l'environnement de production.

TABLE 4.1 – Revue systématique de la littérature

| **Portail de recherche** | **Mots clés** | **Nombre de résultats** |
|---|---|---|
| IEEE Xplore | ((((chaos engineering test software) NOT neural) NOT crypto) NOT image) | 65 (depuis 2004 : 55) |
| researchgate | "chaos engineering software" | 100+ |
| ACM Digital Library | "(+chaos +engineering) AND recordAbstract : (IT cloud microservice software)" | 59 (depuis 2004 : 39) |
| Google Scholar | "Chaos Engineering" test software -neural -encrypt -image | 72 (depuis 2004 : 66) |

#### Résultats

En raffinant manuellement les résultats pour correspondre aux critères de sélection précités, nous obtenons six références. Nous pouvons y ajouter les références croisées des sources initiales pour obtenir un total de neuf. L'analyse statique de celles-ci montre que plus de la moitié sont issues des ingénieurs Netflix et qu'elles sont toutes très récentes (2016 et 2017).
Nous pouvons en déduire que l'ingénierie du chaos est une pratique émergente et confidentielle.

### 4.1.3 Protocole complémentaire

La recherche documentaire donnant un nombre restreint de publications, un protocole complémentaire est ajouté. Il consiste à étendre la recherche par dissémination. En partant des différents auteurs des publications, une exploration est faite à travers divers médias : sites internet, groupes de discussions, conférences et leur diffusion vidéo (YouTube). Chaque nouvel élément donnant naissance à une nouvelle exploration.

#### Validation d'une exploration

Pour être validé comme résultat probant et faire partie du présent document, un élément doit correspondre à tous les critères suivants :

1. avoir comme sujet principal l'ingénierie du chaos ou l'injection de fautes ;
2. avoir au moins un auteur identifiable ;
3. avoir une pérennité suffisante : publication, diffusion vidéo par des organisateurs de conférences reconnus, site internet d'acteurs majeurs (Netflix, Microsoft, etc.).

#### Références internet

Un certain nombre de références proviennent de dépôts publics de sources concernant des projets libres dont il n'a pas été trouvé de publication scientifique. Néanmoins, leurs apports ont été jugés pertinents et font partie intégrante de cette revue.

#### Visualisation des résultats de recherche

Le graphe ci-dessous représente les documents qui sont à l'origine de la revue systématique de la littérature. Les sources principales (*Chaos Engineering* (le livre) Rosenthal et al. (2017), *Chaos Engineering Panel* Basiri et al. (2016), *The Business Case for Chaos Engineering* Tucker et al. (2018), *A platform for automating chaos esperiments* Blohowiak et al. (2016)) sont toutes à l'initiative des pionniers de la discipline travaillant, à l'époque

du moins, chez NetFlix. Le blog technique de cette compagnie (`https://medium.com/netflix-techblog`) contient de nombreuses publications à propos de l'ingénierie du chaos, de ses motivations et de ses outils.

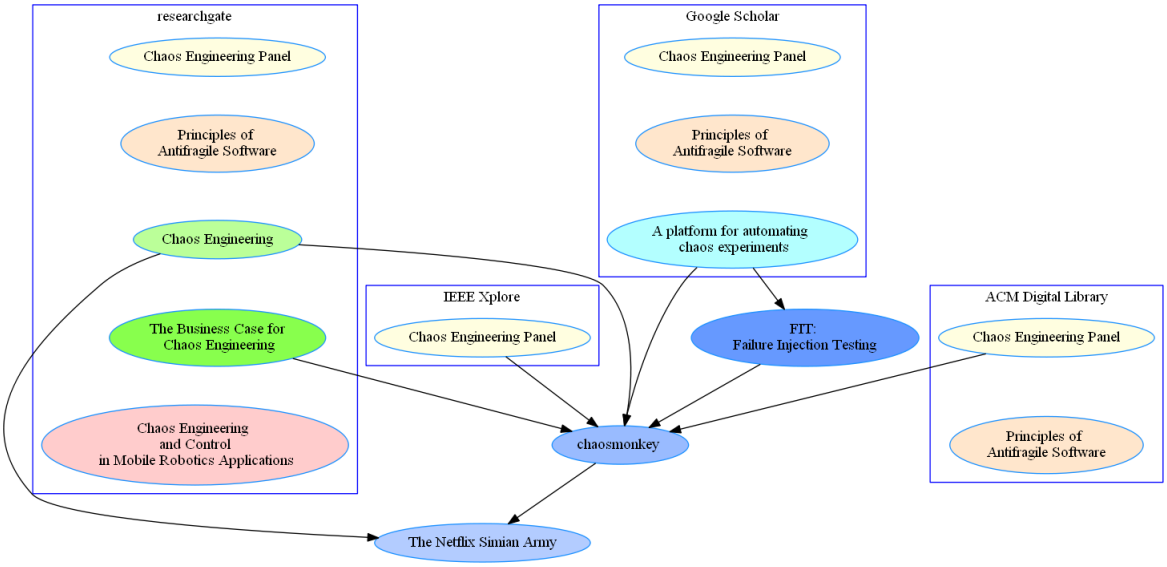


FIGURE 4.1 – Résultats de recherche

Après avoir présenté en détail la méthodologie de recherche et citer les sources principales de l'état de l'art, la section suivante présente les principes qui sous-tendent l'ingénierie du chaos.

## 4.2 Ingénierie du chaos

### Principes de l'ingénierie du chaos

La conception des expérimentations se base sur cinq principes.

1. Construire une hypothèse à propos de l'état stable
   La définition d'un « bon » fonctionnement du système peut être vague. Chez Netflix, les ingénieurs utilisent en premier le nombre de démarrages de vidéo par seconde (SPS, « *starts per second* »). Le SPS varie naturellement et les ingénieurs développent une certaine intuition sur ses variations. Toutes les hypothèses d'une expérimentation s'appuient sur les métriques définies. Les mesures concernent les interactions du système avec l'extérieur. Les évaluations plus fines, telles que l'occupation du CPU, ne sont pas utilisées, car elles ne représentent pas une appréciation de la disponibilité du produit. Elles

sont employées pour évaluer la santé du système qui échappe à la perception des utilisateurs.

2. Sélectionner des éléments réels
   Effectuer des tests logiciels sur les cas nominaux indique que le code fonctionne, pour ces cas seulement. Dans la réalité, des erreurs qui ne font pas partie des tests vont se produire : latences des réseaux, requêtes mal formatées, dysfonctionnement du matériel, etc. Selon Ding Yuan *et al.*, « 92% des échecs catastrophiques proviennent d'une gestion incorrecte d'une erreur non fatale. » Yuan et al. (2014) L'ingénierie du chaos adresse spécifiquement ce genre de problématique en sélectionnant des événements fortuits qui se produisent dans la vie réelle. Certains éléments sélectionnés peuvent concerner une petite partie des utilisateurs et les ingénieurs doivent faire des compromis entre le coût et les risques encourus.

3. Exécuter les expérimentations en production
   Pour des services à la taille d'Internet, la seule solution connue est d'utiliser une architecture distribuée. La nouvelle popularité de l'architecture microservices va amener de plus en plus d'ingénieurs à implémenter des solutions basées sur des systèmes distribués. Malheureusement, les méthodes de test traditionnelles sont incapables d'identifier les dysfonctionnements d'un tel système. Selon la taille de celui-ci, il est impossible de reproduire l'infrastructure dans un autre environnement. De plus, une infrastructure de test ne peut pas reproduire tous les aspects de la production. Les tests doivent donc être réalisés directement sur celle-ci.

4. Automatiser les expérimentations pour les exécuter de manière continue
   Le dernier principe se base sur le maintien de la confiance des résultats sur la durée. Les services sont continuellement mis à jour. Avec toutes les évolutions, la confiance envers les résultats passés s'émousse avec le temps. La première exécution d'une expérimentation se passe manuellement. Mais par la suite, elle est automatisée pour être répétée alors que le système évolue. Chez Netflix, certaines expérimentations s'exécutent continuellement pendant la semaine tandis qu'un large dysfonctionnement (« *Chaos Kong* ») est exécuté mensuellement.

5. Limiter le rayon d'effet
   Tchernobyl, 1986 : le premier accident nucléaire majeur est provoqué par un exercice qui a dégénéré.

Lors d'une expérimentation, il est primordial de garder le contrôle et d'éviter une réaction en chaîne. Pour l'ingénierie du chaos, elle commence donc par un test très limité, habituellement uniquement le poste de l'ingénieur. Ensuite, si l'expérience se passe comme prévu, elle est étendue à un groupe d'utilisateurs. Celui-ci est surveillé et comparé avec un groupe témoin, sur la base de la définition d'un bon fonctionnement. Si l'expérimentation s'y prête, elle est étendue à l'ensemble des utilisateurs.
À tout moment, l'exercice peut être interrompu par une procédure d'urgence, un bouton rouge virtuel. Si, initialement, cet arrêt était une intervention manuelle, il a rapidement fait place à une procédure automatique qui se déclenche dès qu'une anomalie est détectée dans le système.

En se basant su les principes pré-citer, la section suivant décrit brièvement les étapes de réalisation d'une expérimentation.

## 4.3 Exécuter une expérimentation chaotique

Sur base des principes vus précédemment, voici comment se construit une expérimentation :

1. définir l'état stable en prenant quelques mesures du système, dans son comportement normal ;
2. prendre comme hypothèse que l'état stable va se maintenir, autant dans le groupe d'expérimentation que dans celui de contrôle ;
3. introduire des événements du monde réel, tels que le dysfonctionnement de disques durs ;
4. tenter de contredire l'hypothèse en observant les états stables.

S'il suffit de quelques lignes pour présenter la réalisation d'une expérimentation, celle-ci exige néanmoins certains prérequis et un outillage adapté qui sont présenté dans la section suivante.

## 4.4 Prérequis et outillage de l'ingénierie du chaos

Quels sont les prérequis pour adopter l'ingénierie du chaos ? Rapidement, nous pouvons déterminer quelques conditions nécessaires.

1. Le système doit être résilient. S'il est connu qu'il ne peut supporter l'arrêt de services ou qu'il ne peut absorber des piques de latences, c'est qu'il n'est pas prêt pour cette discipline. L'ingénierie du chaos est adaptée pour découvrir de nouvelles faiblesses. Il n'y a pas d'intérêt à mener une expérimentation qui mettrait en avant une faiblesse connue.

2. Le système doit être sous monitoring. L'état du système doit être déterminé pour permettre l'expérimentation. Aucune conclusion ne peut être tirée de celle-ci sans une visibilité sur le comportement du système.

**Modèle de maturité**

Un modèle de maturité est proposé par les auteurs du livre « *Chaos Engineering* » Rosenthal et al. (2017). C'est un modèle basé sur deux axes : adoption et sophistication. L'adoption reflète l'impact que peut avoir l'ingénierie du chaos sur le système d'information tandis que la sophistication reflète la mise en œuvre technologique et l'outillage apportés aux expérimentations.

Une expérimentation menée sans outillage expose le système à de grands risques alors qu'il est difficilement envisageable de mettre en place une solution complexe dès la première expérimentation.

Les valeurs sont proposées dans l'annexe (A).

**Outillage existant**

Injecter de manière contrôlée des fautes dans un système largement distribué demande un outillage approprié. Les ingénieurs de Netflix ont développé plusieurs outils adaptés à leurs environnements. Ils ont ensuite distribué ceux-ci en licence *open-source*. Dans l'article « *Chaos Engineering* » Basiri et al. (2016), les auteurs se demandent si les outils de l'ingénierie du chaos sont liés à une infrastructure particulière et s'il est possible d'en implémenter qui soient réutilisables à travers différents environnements.

### 4.4.1 Chaos Monkey

*Chaos Monkey* (traduction libre :« singe du chaos ») est un projet *open-source* développé par Netflix. Suite à leur migration vers AWS(nuage Amazon), pour éviter les ruptures de production, *Chaos Monkey* a été l'un des premiers services à être délivrés Ciancutti (2010). *Chaos Monkey* est un outil qui, de manière aléatoire, choisit une instance de production et provoque son arrêt. Il s'exécute uniquement les jours ouvrés, sur un système étroitement surveillé.
*Chaos Monkey* a permis aux ingénieurs de découvrir les faiblesses du système, de les adresser et d'augmenter leur confiance envers celui-ci.
Ce produit est passé en version 2.0 en octobre 2016 Lorin Hochstein (2016). Il est maintenant intégré à Spinnaker Netflix (2015), la plateforme de livraison continue de Netflix. Cette intégration permet une plus grande flexibilité

d'utilisation, ainsi qu'une amélioration de l'interface de l'utilisateur.

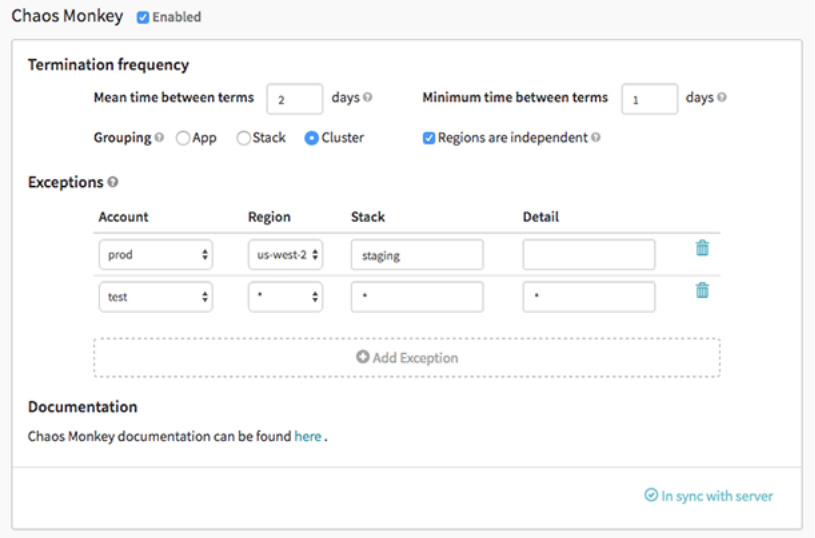


FIGURE 4.2 – *Chaos Monkey*, interface utilisateur dans Spinnaker

### 4.4.2 *Simian Army*

Inspiré par son succès, Netflix a créé différents produits adressant des types d'erreurs particuliers que les ingénieurs ont nommés *Simian Army* (traduction libre : « armée de singes ») Yury Izrailevsky (2011). Ils ont tous comme point commun leur niveau d'intervention. Ils agissent sur l'instance d'un container. L'ensemble des composants (Annexe B) permet à Netflix d'adresser un grand nombre d'erreurs potentielles, de tester la résilience du système et d'augmenter la confiance envers celui-ci.

### 4.4.3 FIT

Les *Monkeys* agissant au niveau de l'instance d'un container, Netflix a développé un autre outil de test du chaos, agissant cette fois au niveau d'un microservice.
L'environnement de production étant particulièrement complexe, il était virtuellement impossible de prédire quels seraient les impacts provoqués, notamment par le *Monkey* de latence. Netflix devait trouver un moyen de limiter les risques des tests, tout en restant dans l'environnement de production.
FIT (*Failure Injection Testing* ; traduction libre : « test d'injection d'erreur ») Kolton Andrus (2014) injecte une simulation d'erreur dans les méta-

données du service de portail, ZUUL. La requête vers un service est décorée et porte ces informations vers celui-ci. La simulation supporte une grande variété de cas : simple délai dans la réponse, système de persistance inaccessible, etc. Chaque niveau d'infrastructure détermine comment émuler, au mieux la demande d'échec de manière réaliste.

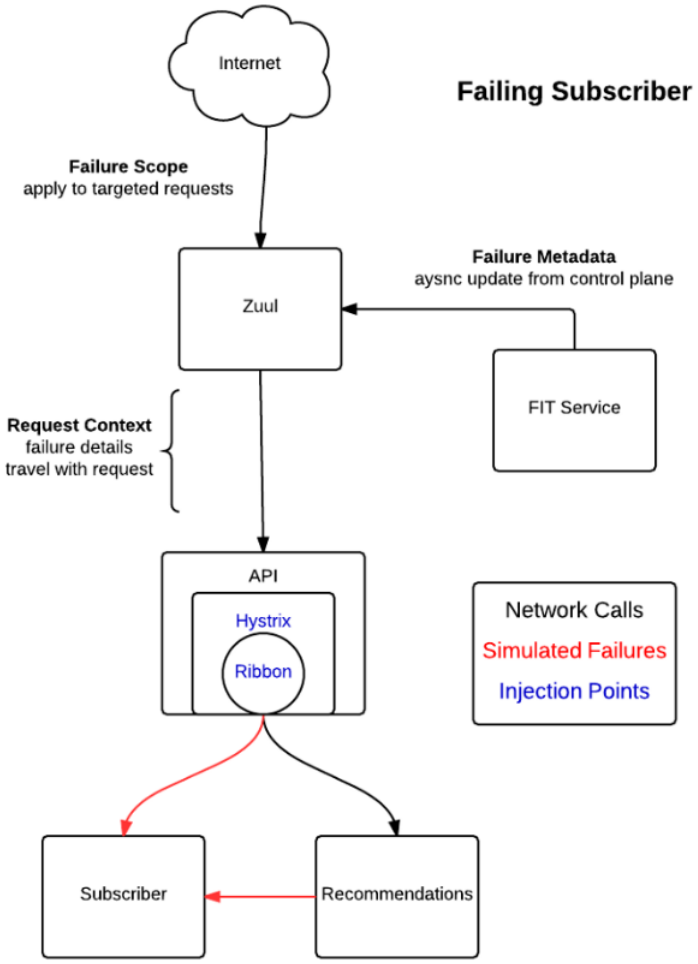


FIGURE 4.3 – FIT, *Failure Injection Testing* (test d'injection d'erreur)

Pour limiter le rayon d'influence du test, FIT utilise les fonctionnalités de ZUUL, un service de portail qui fournit un routage dynamique, du monitoring, de la résilience, de la sécurité, etc., pour inspecter et gérer le trafic. Pour la majorité des tests, ZUUL est utilisé pour limiter leurs effets à des comptes de test ou à certains appareils. Si le test est concluant, il est graduellement étendu à l'ensemble des requêtes.

### 4.4.4 Chap

Si FIT permet un contrôle sur le rayon d'action du test, son mécanisme de mesure ne permet pas un contrôle fin sur le système. Agissant sur l'ensemble des requêtes passant par le portail ZUUL, un test peut influer sur un nombre significatif de clients, sans pour autant lever un avertissement. Il est très difficile de faire la différence entre un test chaotique et un bruit du système de surveillance.

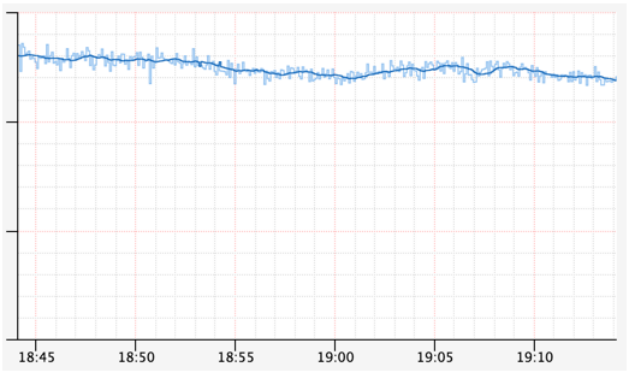


FIGURE 4.4 – Expérimentation avec FIT

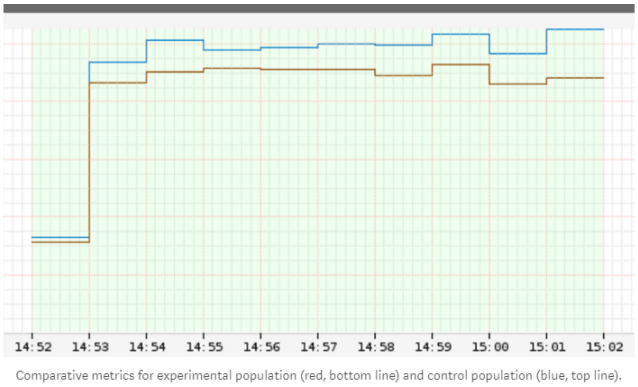


FIGURE 4.5 – Expérimentation avec ChAP

ChAP Blohowiak et al. (2016) utilise la technique de témoins ou objets de références Pierre (2012) qui consiste à comparer les observations d'une expérience avec les observations d'un groupe non affecté par celle-ci.

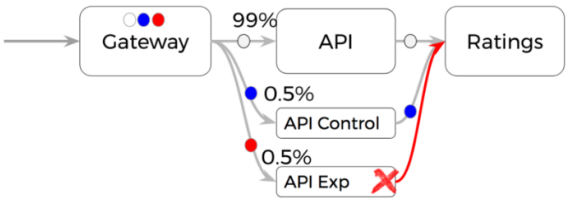


FIGURE 4.6 – ChAP, principe d'expérimentation

### 4.4.5 Gremlin

Gremlin est un produit logiciel commercial qui permet de mettre en place l'ingénierie du chaos. Il est présenté par ses fondateurs comme un *framework* permettant de simuler des pannes de manière sûre, sécurisée et facile. Il permet de réaliser des expériences chaotiques, d'identifier les points faibles du système et de les fixer avant qu'ils ne deviennent un problème Inc. (2016). Gremlin doit s'installer sur chaque machine cible. Selon sa documentation, le produit est disponible pour les systèmes Linux ou le container Docker.

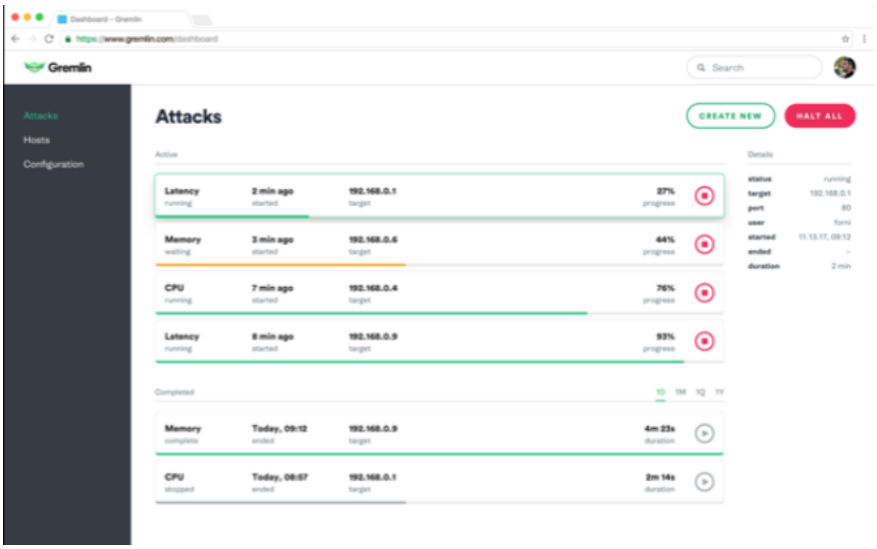


FIGURE 4.7 – Gremlin, interface utilisateur

Cette section a effectué une revue des prérequis et de l'outillage nécessaires à la réalisation d'une expérimentation.
Il existe cependant des variantes à l'approche initiale. La section suivante les présente brièvement.

## 4.5 Ingénierie du chaos, variantes

**Azure préproduction, pattern de recherche**

Dans un article de blog publié sur *azure.microsoft.com*, Heather Nakama ingénieur logiciel, explique comment l'ingénierie du chaos est utilisée dans *Azure Search* Nakama (2015).
Microsoft a développé son propre outil nommé « *Search Monkey* » (« singe de recherche », en référence aux *singes* de Netflix), un service de recherche d'anomalies pour son environnement Azure (nuage Microsoft) de pré production. Ce service change aléatoirement la topologie et l'état de l'environnement. Il est utilisé pour détecter des anomalies ou des faux positifs dans le système de monitoring. Ce service a permis de détecter plusieurs anomalies, avant d'atteindre la production.
Le service tourne continuellement. Les opérations possibles qu'il peut tester, dépendent du niveau de chaos sélectionné.

Le niveau *bas* concerne les erreurs que le système peut récupérer, sans interruption de service ou avec une interruption minimale. Dans ce niveau, si une alerte est détectée par le test, elle est considérée comme une anomalie.
Pour le niveau *médium*, les erreurs peuvent aussi être récupérées par le système, mais dans un mode dégradé, affectant ses performances ou ses disponibilités. Une alerte de basse priorité peut être lancée sur le système de monitoring.
Le niveau *haut* est le niveau le plus catastrophique et provoque des interruptions de service. Des alertes de hautes priorités sont lancées aux personnels de garde et des opérations manuelles sont nécessaires pour réparer le système. Ce niveau de test est important car il permet de vérifier si, lors d'une panne majeure, le système peut être récupéré tout en garantissant l'intégrité des données des clients.
Tous les tests, quel que soit le niveau sélectionné, peuvent être exécutés à la demande.
L'approche vise à rétrograder des erreurs de exceptionnel à attendu. Les différents niveaux de test fournissent une approche systématique et itérative. Une erreur détectée peut alors être incorporée comme erreur potentielle attendue dans l'infrastructure et des mesures peuvent être prises pour y réagir au mieux. Les erreurs peuvent être des cas théoriques, des anomalies détectées par une expérimentation chaotique ou des erreurs reproduites suite à un problème réel de production.

Il existe un dernier niveau d'erreur, le niveau *extrême*. Dans ce cas, la perte de donnée est possible ou le système peut échouer silencieusement, sans aucune alerte. Il est impossible de prévoir l'état du système après une expérimentation de ce genre. C'est pourquoi, elle n'est jamais lancée de manière aléatoire ou continue. Lorsque l'opération peut être abaissée au niveau *haut*, elle est incluse dans l'automatisation.
Le travail sur le système consiste à réduire le plus possible le niveau de chaque événement. L'expérimentation chaotique, menée de manière automatique, permet de découvrir des failles de degré d'importance divers, d'adresser les plus pertinentes de manière planifiée et d'augmenter ainsi les disponibilités des services en production.

**Environnement de reprise après sinistre (*Disater Recovery*)**

Lors d'un conférence « *O'Reilly Velocity, Too Big to Test : Breaking a production brokerage platform without causing financial devastation* » Parrish and Halsey (2015) (traduction libre : « Trop grand pour être testé : casser un système de courtage de production sans causer de désastre financier »), Kyle Parrish et Dave Halsey, ont expliqué leur approche pour maîtriser le chaos.
En préambule, nous pouvons remarquer que les deux auteurs sont des dirigeants dans une banque d'investissement, *Fidelity Investments*. Kyle Parrish est directeur des risques technologiques tandis que Dave Halsey est vice-président de l'assurance qualité. Nous quittons également le monde des grands acteurs d'Internet pour celui de la finance, leur employeur gérant pas moins de 5,4 milliards de dollars d'avoirs.

Les marchés financiers sont hautement volatiles. Le 6 mai 2010, se produit à la bourse de New-York le « *Flash Crash* », « *Krash éclair* », un effondrement boursier rapide, une baisse de 9,2% en dix minutes.
Les banques se doivent de protéger les intérêts de leurs clients, en possédant des systèmes d'information capables de résister à l'imprévisible.
Un système d'échange boursier est une solution complexe. Dans le cas de *Fidelity Investments*, son architecture s'articule autour d'un *Mainframe*. Pour des raisons financières et légales, il n'est pas envisageable d'utiliser l'environnement de production pour effectuer des expérimentations.

Les ingénieurs ont décidé d'utiliser l'environnement de reprise après sinistre (*Disater Recovery, DR*). Il a l'avantage d'être dimensionné pour reprendre la production à tout moment et possède les données nécessaires.
Pour mener à bien ce projet, les efforts sont répartis comme suit : la compréhension du système, la compréhension des données et l'évaluation permanente des risques. Le travail demande la coordination de nombreuses personnes dans une entreprise complexe.

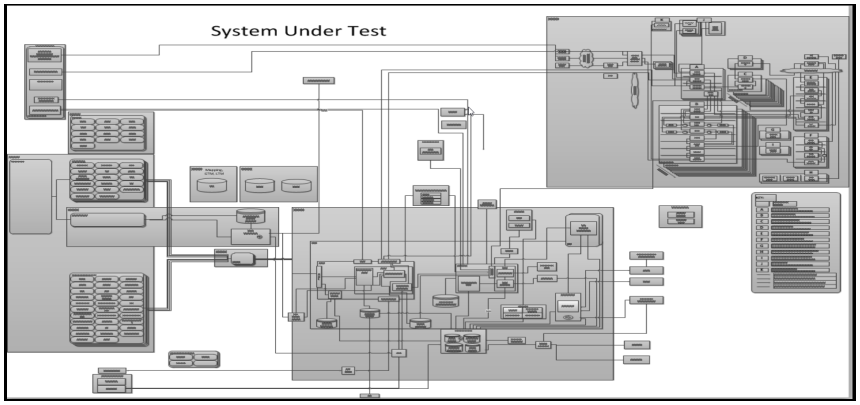


FIGURE 4.8 – Complexité d'un système d'échange boursier (source O'Reilly Velocity)

Les résultats sont concluants. Le système s'adapte plus facilement tandis que ses limites sont connues.

Cette section termine la présentation détaillée de l'ingénierie du chaos. Mais comme toute discipline, elle possède ses propres limites qui sont présentées dans la section suivante.

## 4.6 Limites de l'ingénierie du chaos

L'ingénierie du chaos est une discipline qui permet de découvrir des failles dans un système complexe et d'apprendre un peu plus le fonctionnement de celui-ci. Malheureusement, la plupart des techniques possèdent leurs propres failles et limites.

### Correction des failles

Trouver une faille dans un système, complexe ou non, peut être une tâche difficile. Sa correction peut être risquée et beaucoup plus compliquée. Une erreur détectée peut révéler un problème conceptuel qui invalide l'architecture logicielle mise en place.
Un des avantages des architectures microservices est l'agilité des équipes de développement pour fournir rapidement de nouvelles fonctionnalités aux utilisateurs. Corriger une erreur qui, potentiellement seulement, peut amener à un système plus fragile n'est pas toujours considéré comme une nouvelle valeur pour le système. Selon Warner Richard et Sloan Robert, la stratégie typique du développement de logiciel est de garder les coûts au plus bas et d'être le premier sur le marché, même si cela signifie de fournir un produit comportant des vulnérabilités significatives Warner and Sloan (2011).

### Gestion des risques

Un produit logiciel qui apporte ou supporte une activité peut être considéré comme n'importe quel outil ou unité de production. Une stratégie de gestion pour une entreprise consiste à gérer les risques liés à cette unité.
Si le produit logiciel est critique, au point qu'une altération puisse entraîner une perte de revenu significative, chaque modification volontaire sera probablement analysée avec soin. L'apport constant de nouvelles fonctionnalités est considéré comme vital pour le métier. Il n'en est pas de même pour des événements qui sont capables de perturber négativement le produit, sans apporter la moindre nouvelle fonctionnalité.
Une des limites de la discipline chaotique est cette capacité à devoir démontrer en permanence que les bénéfices de la pratique sont supérieurs aux risques supportés.

### Personnel hautement qualifié

Selon Alvaro, imaginer, concevoir, implémenter et exécuter une expérimentation chaotique reste un travail qui exige du personnel extrêmement qualifié. L'ingénieur du chaos est une sous-classe hautement spécialisée de l'ingénieur en fiabilité. Il faut également la collaboration de diverses équipes pour comprendre les singularités des composants et leurs interactions pour implémenter une injection d'erreur. L'ingénieur du chaos doit ensuite cibler un service qui lui semble avoir une faiblesse en tolérance d'erreur.
Non seulement cette approche est difficile et doit être répétée à chaque nouveau composant, mais le modèle de conception est entièrement caché dans le cerveau de l'ingénieur. Elle s'apparente à de la prêtrise avec ses icônes (les tableaux de bords) et ses rituels (manuels) Alvaro and Tymon (2017).
L'intervention du génie humain est une faiblesse majeure de la discipline, rendant l'approche fragile et non évolutive.

### Paradoxe de l'automatisation de l'injection de fautes

Lorsqu'une faute est injectée automatiquement, aléatoirement et absorbée de manière transparente par le système, elle passe totalement inaperçue. C'est le but recherché par sa conception : mettre en évidence les faiblesses du système et renforcer la confiance en celui-ci. Mais si des fautes sont générées par le système lui-même, le monitoring ne lancera une alerte qu'à la condition qu'un service soit défectueux ou indisponible.
L'injection de fautes devient alors un vecteur de fragilité. Non seulement il masque des problèmes potentiels, mais il peut produire une suite combinatoire inédite, fatale pour le système.

**Augmentation de la complexité**

La section sur l'outillage chaotique présente les idées et le fonctionnement général des outils. Elle n'a pas pour vocation d'expliquer en profondeur comment les composants sont capables de modifier le système pour provoquer une faute. La tâche n'est pas simple. Pour certains outils, comme ChAP, ils sont intimement liés à l'environnement de production.
De plus, la possibilité d'injecter des erreurs doit être contrôlée et soumise à des droits sur le système.
Tous ces éléments viennent donc ajouter de la complexité à un système qui est en constante évolution.

**Renforcement et immunité : infinie variabilité**

L'ingénierie du chaos peut être comparée à l'utilisation d'un vaccin. Le vaccin est une agression contrôlée d'un organisme afin de renforcer son système immunitaire. Cet organisme est ensuite résistant aux attaques couvertes par ce vaccin. Sa résilience n'a pas augmenté et il reste non préparé face aux innombrables autres agressions.
Tout comme un vaccin, l'ingénierie du chaos permet d'augmenter l'espérance de vie du système. Elle augmente la probabilité que le système puisse survivre. Celui-ci, évoluant en permanence, produit par l'explosion combinatoire une variabilité infinie d'événements potentiellement perturbateurs. L'ingénierie du chaos ne peut lui offrir une quelconque immunité.

Cette section vient de présenter les limites de l'ingénierie du chaos. Si certaines sont exogènes, comme la difficulté de correction d'une faille, d'autres sont intrinsèques à la discipline, comme le manque de scalabilité du à l'utilisation d'un personnel très qualifié. La section suivante présente le LDFI, une approche différente qui tente d'adresser cette limitation.

## 4.7 Lineage-Driven Fault Injection (LDFI)

Si l'ingénierie du chaos permet d'augmenter la compréhension d'un système largement distribué et de découvrir certaines de ses faiblesses, c'est une discipline qui a une scalabilité limitée due à son besoin de personnels très qualifiés et qui ne peut pas s'affranchir de la complexité du produit.

### 4.7.1 LDFI

Pour s'affranchir du processus essentiellement humain de l'ingénierie du chaos, Alvaro décrit dans sa thèse comment il a adapté et implémenté un prototype de recherche, appelé « *lineage-driven fault injection* » (LDFI, traduction libre : « *injection de fautes conduite par lignée* ») pour automatiser

les expérimentations chaotiques chez Netflix Alvaro et al. (2016).
Tester tous les scenarii est trop coûteux. Utiliser une exploration aléatoire est un gaspillage de ressources pour tous les cas jugés inintéressants. Se baser sur l'approche des ingénieurs donne des résultats à la hauteur de l'intuition de ces derniers, tout en étant peu évolutif.
LDFI est une technique pour guider les recherches à travers les injections de fautes possibles. Il se base sur deux idées. La première est qu'un système est tolérant à la panne s'il fournit suffisamment d'alternatives pour délivrer le comportement attendu. Si nous possédons de manière précise les informations à propos de ces alternatives, nous pouvons déterminer pour quelles fautes le système est tolérant et inversement dans quels scénarios il est fragile. La seconde idée est que, au lieu de partir de l'état initial du système et d'exhaustivement rechercher l'espace de possibilités d'exécutions, une meilleure stratégie pour identifier rapidement les problèmes de tolérance est de partir du résultat et de raisonner à rebours, de l'effet vers la cause, pour comprendre quelles combinaisons de fautes sont à éviter.
Voici les étapes du LDFI :

1. en partant de la réponse finale, il capture les données concernant, à chaque étape, *pourquoi* le système fournit cette réponse. Il construit un graphe de lignées qui caractérise les données et les calculs qui ont contribué au résultat ;
2. sur la base du graphe, en est déduit un ensemble distinct de chemins conduisant de l'appel initial vers le résultat final. Chaque chemin est une alternative suffisante. Pour chacun d'eux, sont encodés des propositions booléennes représentant les évaluations d'erreur sur des nœuds invalidant tout le chemin ;
3. un proposition n'indique pas forcément un problème de tolérance, mais doit être vérifiée. Une injection de fautes correspondant à la proposition est introduite. Soit le système est en erreur, soit de nouvelles données de lignées sont ajoutées au graphe ;
4. pour terminer, LDFI certifie que, pour la configuration et les informations données, le système est tolérant à la panne.

LDFI a donné des résultats encourageants, mais son implémentation dans un système réel a été délicate. Pour commencer, un système de production présente un ensemble de contraintes qui lui sont propres. Il n'est, par exemple, pas possible de rejouer des requêtes non idempotentes. Ensuite, le système existant s'est révélé bien plus complexe que le modèle. Modifier le système était risqué tandis que complexifier le modèle n'était pas désirable, sa force étant sa simplicité et son abstraction.

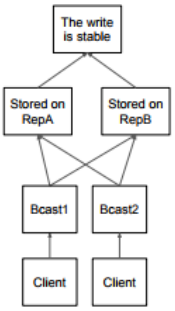


FIGURE 4.9 – Graphe LDFI, sourceAlvaro et al. (2016)

### 4.7.2 Critique du LDFI

Le LDFI soulève plusieurs critiques. Pour commencer, il demande des connaissances en programmation peu présentes sur le marché de l'emploi. Ensuite, il exige une spécification détaillée du système, alors que les ingénieurs du chaos déclarent ne pas les posséder. Pour terminer, son mécanisme le confine aux appels idempotents.

#### Programmation

L'implémentation du LDFI Alvaro et al. (2015) utilise le langage *Dedalus* comme source d'entrée. *Dedalus* est une variante du *Datalog*, lui-même un sous-ensemble de *Prolog*, un langage de programmation logique. Une entrée doit décrire précisément l'environnement à vérifier. Chaque déclaration est une expression booléenne de la forme :

```
conclusion(bindings1)[@annotation] :-
    premise1(bindings2),
    premise2(bindings2) [...],
    notin premisen(bindings3), [...];
```

L'exemple suivant est tiré du fichier d'introduction de *Molly* :
« La règle suivante indique que pour chaque paire d'enregistrements dans bcast et node qui correspondent dans leur première colonne, il devrait (à un moment inconnu) créer un enregistrement dans le journal [...]. Intuitivement, cela capture la communication en multidiffusion : quand node1 a un enregistrement, pour chaque node2 connu, il transmet le contenu de cet enregistrement à Node2.

```
log(Node2, Pload)@async :-
    bcast(Node1, Pload),
    node(Node1, Node2);
```

» Alvaro et al. (2015).

La nécessité de fournir la description du système est la première critique que l'on peut émettre à propos de l'utilisation de *Molly* dans un système réel. Ce système peut ne pas être connu. Son implémentation effective peut être différente de la connaissance des ingénieurs. La critique suivante vient du langage d'entrée utilisé. Les langages de programmation logique sont certes enseignés dans les universités, mais ils sont peu utilisés dans l'industrie du logiciel, à l'exception du domaine de l'intelligence artificielle. Une recherche menée sur le réseau social professionnel *LinkedIn*, concernant *Prolog* dans le monde entier, donne 2 050 offres d'emploi (une seule offre pour la Belgique, en intelligence artificielle). La même recherche avec *Java* propose 371 000 offres (444 000 pour SQL).

**Spécifications**

*Molly* apporte la preuve de l'exactitude du système ou du manquement à celle-ci. Pour apporter cette preuve, il faut lui fournir les spécifications précises de tous les composants, sous la forme de préconditions et postconditions dans le langage Dedalus.
Exemple des auteurs de *Molly* : «

```
pre(X, Pl) :-
    log(X, Pl),
    notin bcast(X, Pl)@1,
    notin crash(X, X, _);
post(X, Pl) :-
    log(X, Pl),
    notin missing_log(_, Pl);
missing_log(A, Pl) :-
    log(X, Pl),
    node(X, A),
    notin log(A, Pl);
```

» Alvaro et al. (2015).

La nécessité de fournir les spécifications précises des composants du système, aggrave les critiques concernant la description de celui-ci. L'expert dans le langage *Dedalus* devra obtenir cette dernière de manière complète et détaillée, tout en espérant qu'elle est correcte et actualisée.
La réalisation de la programmation et de la spécification, nécessaire à chaque mise à jour du système, demande un temps non négligeable. De plus, étant elle-même une tâche de programmation, elle est sujette à l'erreur et devrait être vérifiée. Molly peut apporter la preuve de l'exactitude d'un système, à condition que les ingénieurs apportent la preuve de l'exactitude de leurs réalisations, programmation et spécification du système, en *Dedalus*.

**Idempotence**

Un système LDFI utilise l'injection de fautes en production pour découvrir la robustesse du système. Si une faille potentielle est découverte, une injection est générée, pour une même requête. Dans un système réel, seuls les éléments idempotents peuvent supporter ce mécanisme. On peut difficilement envisager un système bancaire qui effectuerait plusieurs fois le même transfert d'argent. Il faut donc une connaissance fonctionnelle précise et détaillée du système pour sélectionner les composants qui supportent le LDFI.

### 4.7.3 Conclusion

Le LDFI apporte la formidable faculté de prouver la correction d'un système, ce qui est particulièrement précieux pour la vérification d'un nouveau protocole de communication dans un système largement distribué. Les auteurs de l'implémentation Molly ont démontré que cela fonctionnait sur des systèmes réels.
Il exige cependant les éléments suivants :

1. Utiliser du personnel très qualifié connaissant un langage peu répandu ;
2. Obtenir les spécifications de chaque composant du système ;
3. Obtenir la description de l'implémentation actuelle du système ;
4. Obtenir la connaissance fonctionnelle détaillée du système ;

Si tous les éléments ci-dessus ne sont pas la compétence d'une même personne, cela nécessitera des ressources de coordination supplémentaires.

Cette section a présenté le LDFI et les critiques que l'on peut lui émettre à son égard. Elle servira de base pour l'approche par rétro-ingénierie, détaillée dans la partie suivante de ce mémoire.
Le chapitre suivant présentera d'abord les éléments théoriques nécessaires à cette partie.

# Chapitre 5

# Rétro-ingénierie

Ce chapitre présente les éléments théoriques utilisés dans la deuxième partie de ce mémoire :

— La rétro-ingénierie ;
— La récolte d'informations à propos d'un système ;
— La découverte du système, avec une approche bayésienne ;
— La visualisation d'un système ;
— L'injection de faute ;
— L'évaluation des résultats.

## 5.1 Méthode de rétro-ingénierie

Dans l'article « Understanding database schema evolution : A case study » Cleve et al. (2015) (traduction libre : Comprendre l'évolution du schéma de base de données : une étude de cas), la rétro-ingénierie, par l'analyse SQL, propose « une méthode assistée par un outil pour analyser l'historique de l'évolution des bases de données existantes, et nous rapportons une étude de cas à grande échelle sur la rétro-ingénierie d'un système d'information complexe. » (Ibid.)
La méthode consiste en : (1) l'extraction du code SQL dans le gestionnaire de source ; (2) l'extraction du schéma logique ; (3) la comparaison des schémas logiques ; (4) la visualisation et l'exploitation des schémas.

D'autres méthodes de rétro-ingénierie sont utilisées pour découvrir le fonctionnement d'un logiciel : la lecture et la compréhension du code source, la décompilation du code binaire ou intermédiaire, l'interprétation des messages inter processus, l'analyse des trames réseau, etc.

La méthode de rétro-ingénierie sélectionnée dans la prochaine partie est la suivante : (1) récolte d'informations dans les logs ; (2) inférence à propos du système ; (3) visualisation du système découvert ; (4) injection de fautes ;

(5) évaluation des résultats.

Les sections suivantes présenteront les différents éléments théoriques utilisés dans la méthode précitée.

## 5.2 Récolte d'informations

### 5.2.1 Information disponible

La récolte de l'information est la première étape de la rétro-ingénierie. Il peut s'agir de codes sources, de codes binaires, de messages de différents formats, de logs réseau, d'accès ou applicatifs. Les informations doivent être suffisantes pour permettre la compréhension, parfois partielle, du fonctionnement interne du système.
De la qualité et de la couverture de l'information dépendra le succès de la rétro- ingénierie.

### 5.2.2 Principes de moindre ingérence

Pour récupérer les informations, il existe plusieurs techniques. Dans le contexte de l'ingénierie du chaos, il s'agit d'environnements complexes. Une des faiblesses identifiées est l'augmentation de la complexité. Pour limiter ce défaut, la récolte d'informations dans ce cadre doit être la moins perturbante possible. Dans le meilleur des cas, les informations sont récoltées sans modifier le système.
Les techniques suivantes sont retenues pour la rétro-ingénierie : la journalisation des données (log data) et l'instrumentalisation.

#### Journalisation des données

Dans leur livre « Utilisation du pare-feu et des journaux d'application » les auteurs déclarent « Les logs peuvent être un endroit important pour obtenir des informations [...]. Tous les systèmes de journalisation incluent différents formats pour les fichiers de log. L'analyse des logs peut avoir des besoins différents, mais l'un des avantages des fichiers journaux est qu'ils ne sont souvent basés que sur du texte. » Joh (2017)

#### Instrumentalisation

Lorsqu'une application ne fournit pas de logs exploitables pour la rétro ingénierie, il est possible de produire une certaine quantité d'information en instrumentalisant des composants d'infrastructure, généralement présents dans une architecture utilisant les microservices.
Pour Montesi and Weber (2016), « les patrons les plus largement utilisés pour la programmation de microservices » sont les « : disjoncteur [circuit

breaker ,NDT], découverte de service et passerelle API [API Gateway, NDT] » (traduction libre).
À l'instar de ce qui a été réalisé dans l'implémentation de LDFI chez Netflix Alvaro et al. (2016), la passerelle API est le composant idéal pour réaliser une instrumentalisation.

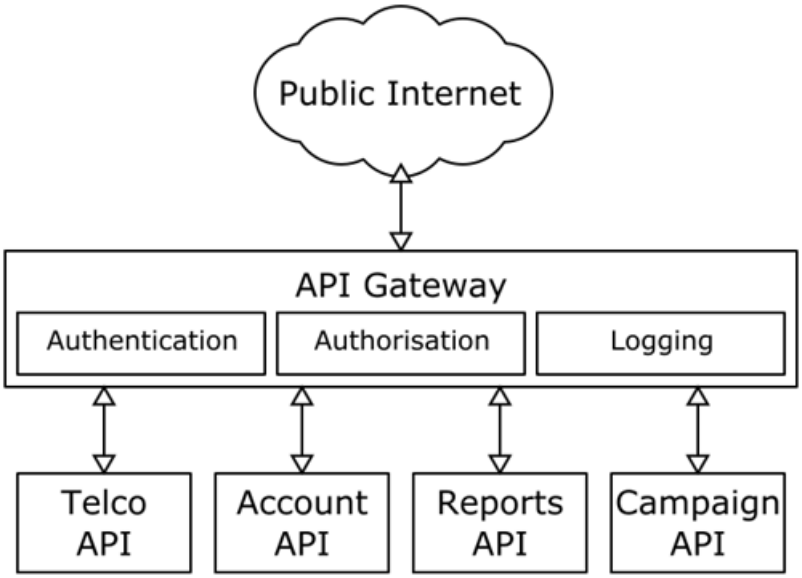


FIGURE 5.1 – Passerelle API Russell (2017)

Selon Russell (2017), « positionnée en tant que reverse proxy entre les clients et les API, la passerelle API est très bien située pour fournir une journalisation cohérente sur tous les points de terminaison de service. Cela ne veut pas dire que les services d'arrière-plan n'ont pas besoin de consigner leur comportement - ils le devraient - mais la passerelle API peut fournir une capacité de consignation uniforme et par défaut pour tous. »

## 5.3 Découverte du système

Dans un système largement distribué, constitué de nombreuses instances de microservices, une des tâches de la rétro-ingénierie consiste à identifier deux services ou fonctionnalités qui sont fonctionnellement identiques. Pour y parvenir, une méthode possible est l'utilisation d'inférences bayésiennes. Cette section présente le théorème de Bayes et son application

### 5.3.1 Approche bayésienne

Deux fonctionnalités sont identiques si elles fournissent le même résultat pour de mêmes entrées. Elles peuvent cependant emprunter des chemins différents et apparaître comme différentes.
La découverte du système doit deviner quelles sont les fonctionnalités identiques.

#### Théorème de Bayes

Dans son livre « La Formule Du Savoir, une philosophie unifiée du savoir fondée sur le théorème de Bayes » Nguyên Hoang présente le théorème de Bayes, un théorème mathématique découvert par Thomas Bayes et retrouvé indépendamment par Pierre-Simon Laplace. « [...] la théorie statistique de la décision montre que les inférences bayésiennes sont essentiellement les seules méthodes d'apprentissage admissibles, dans le sens où une méthode n'est jamais dominée par une autre, si et seulement si, elle revient à appliquer la formule de Bayes » Hoang (2018)

La formule de Bayes : $P(T|D) = \frac{P(D|T)P(T)}{P(D)}$

Soit, $P(X|Y)$ pour probabilité de $X$ en connaissant $Y$, appelé également loi a posteriori :
la probabilité de la théorie T en connaissant les données D est égale à :
la probabilité d'avoir les données $D$ en sachant la théorie $T$ (la vraisemblance),
multipliée par la probabilité d'obtenir la théorie $T$ (loi a priori),
divisée par la probabilité d'obtenir $D$.

#### Approximation de Bayes

L'application stricte de la formule de Bayes, le *bayésianisme pur*, demande des temps de calcul qui peuvent être extrêmement longs.
Dans le même livre Lê Nguyên Hoang (ibid.) propose cinq approches pour passer d'un bayésianisme pur à un bayésianisme pragmatique.

«Une première approche consiste à considérer uniquement un nombre restreint de modèles candidats. [...] Cette approche pourrait être typiquement complétée par l'algorithme par *multiplicative weights update*. [...]
Une deuxième approche consiste à ne calculer qu'un modèle à forte crédence(SIC), voire à identifier le modèle le plus crédible. C'est ce que l'on appelle la maximisation de l'a posteriori (MAP). [...]
Une troisième approche consiste à ignorer la fonction de partition, ce dénominateur de la formule de Bayes qui requiert la comparaison de tous les modèles imaginables. [...]
La quatrième approche que j'aimerais mentionner est la plus étrange. Elle consiste à s'autoriser de violer les lois des probabilités.[...] Cette largesse semble conduire à des résultats qui ont l'avantage d'être rapides à calculer. [...]
Enfin une cinquième et dernière approche consiste à considérer un ensemble restreint de lois de probabilité. De façon cruciale, il est alors important de disposer d'une mesure de similarité entre des lois de probabilité. »

### 5.3.2 Approximation de Bayes appliquée

Les approches pour effectuer une approximation de Bayes pragmatique sont également présentes dans le domaine de l'intelligence artificielle. La méthode des poids multiplicatifs (multiplicative weight update method en anglais) est un algorithme qui peut être implémenté avec TensorFlow, l'outil open source d'apprentissage automatique développé par Google, tandis que la sélection d'un modèle à fort crédit est effectuée par des algorithmes tels que « expectation-maximization (EM) ou les generative adversarial networks (GANs) » (ibid.).

Cette section a présenté les éléments théoriques permettant de découvrir un système donné. La section suivante présente deux manières de représenter visuellement ce résultat.

## 5.4 Visualisation des résultats

### 5.4.1 Vizceral

Pour la visualisation de leur large système, Netflix a développé Vizceral. Selon leurs auteurs « Vizceral est né de la nécessité de pouvoir comprendre l'état de l'architecture de microservices complexe de Netflix dans une région donnée, en un coup d'œil, lors d'un basculement de trafic. [...] En ce qui concerne les métriques elles-mêmes, étant donné que nous traitons avec des valeurs métriques trop grandes pour pouvoir visualiser 1 : 1 efficacement, il existe un certain niveau d'échantillonnage dans le nombre de points animés. Lorsqu'il y a tant de choses à suivre et à surveiller, les chiffres exacts cessent d'être importants et les informations relatives sont beaucoup plus exploitables. » Reynolds et al. (2017)

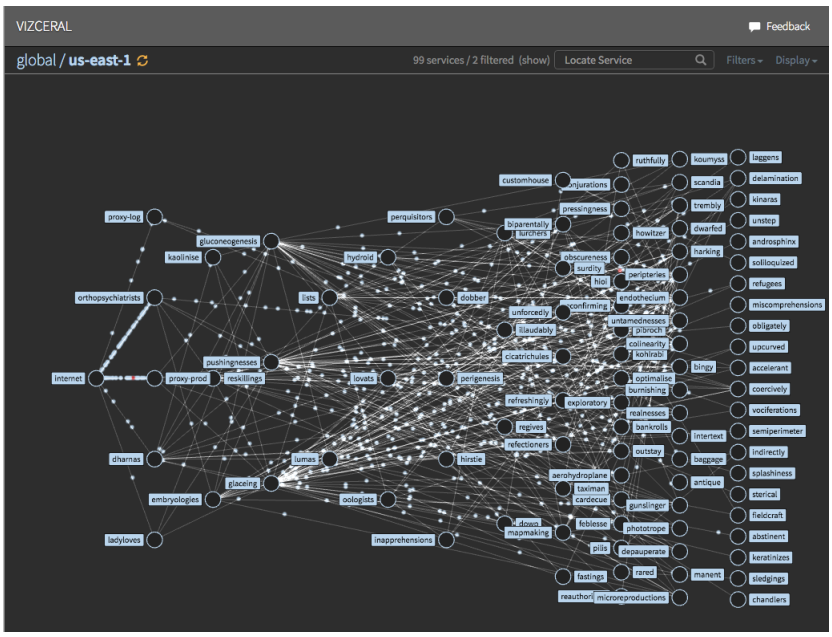


FIGURE 5.2 – Vizceral. Reynolds et al. (2017)

Dans leur article « Flux : une nouvelle approche de l'intuition du système (Flux :A New Approach to System Intuition en anglais) » les ingénieurs de Netflix expliquent : « Dans les équipes Trafic et Chaos de Netflix, nous voulons comprendre en temps réel l'effet d'une variable sur un sous-ensemble de trafic de requêtes au cours d'une expérience Chaos. Nous avons besoin d'un outil capable de nous donner cette compréhension globale du trafic dans notre système complexe et distribué. » Kosewski et al. (2015)

### 5.4.2 Graphviz

« Graphviz est une collection de logiciels permettant de visualiser et de manipuler des graphiques. Il fournit une visualisation graphique pour les outils et les sites Web dans des domaines tels que logiciels, réseaux, bases de données, représentation des connaissances, et bio-informatique. [...] De nombreuses applications utilisent Graphviz pour produire des présentations graphiques afin d'aider leurs utilisateurs à mieux comprendre les informations de domaine [...] » Ellson et al. (2003)

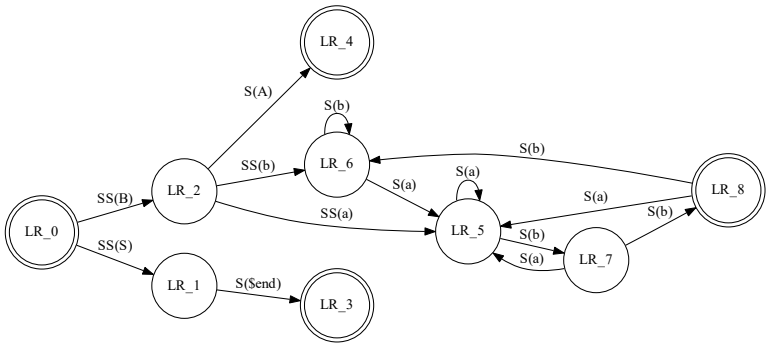


FIGURE 5.3 – Graphviz (graphviz.gitlab.io)

## 5.5 Injection de fautes

### 5.5.1 Outils d'injection de fautes

Dans cette section nous allons passer en revue quelques outils d'injection de fautes qui semblent pertinents pour la rétro-ingénierie. La caractéristique principale recherchée sera la moindre ingérence dans le système qui par définition est complexe et probablement pas totalement maîtrisé.

#### Armée de singes

Quand on parcourt la littérature de l'ingénierie du chaos, un des outils d'injection de fautes qui se présente naturellement est le fameux *Simian Army* (armée de singes) de Netflix. Développé sur mesure, *Chaos Monkey* (singe du chaos) est utilisable en dehors de l'infrastructure de Netflix. Selon la documentation du code source, « Chaos Monkey est entièrement intégré à Spinnaker, la plate-forme de distribution continue que nous utilisons chez Netflix. Vous devez gérer vos applications avec Spinnaker pour pouvoir utiliser Chaos Monkey pour mettre fin aux instances.
Chaos Monkey devrait fonctionner avec tous les backends compatibles avec Spinnaker (AWS, Google Compute Engine, Azure, Kubernetes, Cloud Foundry). Il a été testé avec AWS, GCE et Kubernetes. » Hochstein and open source (2017)
« Spinnaker est une plate-forme de diffusion continue, multi-cloud, à code source ouvert permettant de publier des modifications logicielles avec une grande rapidité et une grande confiance. » Netflix (2015). Si le système utilise cet outil de diffusion continue, Chaos Monkey est probablement l'outil le plus pertinent, notamment la version de base qui termine un conteneur et celle qui introduit une latence.

#### Zuul

« *Zuul* est un service de périphérie qui assure le routage dynamique, la surveillance, la résilience, la sécurité, etc. » Macero (2017) *Zuul* implémente le patron de passerelle d'API (API gateway pattern, en anglais) qui consiste à placer un point d'entrée unique pour les clients d'un système possédant de nombreux points terminaux. Le cœur de *Zuul* est un ensemble de filtres agissant tant sur les requêtes que sur les réponses.
À l'instar de Chaos Monkey, le système doit cependant utiliser *Zuul* comme composant de son architecture distribuée. Un autre outil remplissant le rôle de portail d'API pourrait probablement remplir cette tâche.

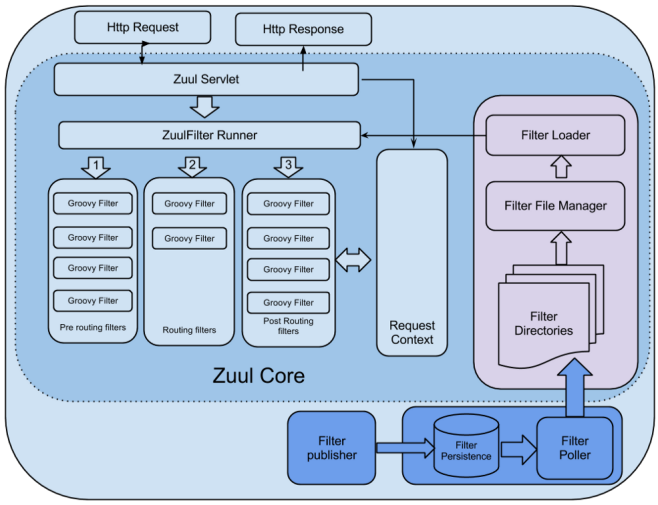


FIGURE 5.4 – Zuul Macero (2017)

### Spring Boot

Pour les microservices écrits avec le langage Java et le framework Spring-Boot, déployé dans un environnement n'utilisant pas de Spinnaker, la librairie chaos-monkey-spring-boot est disponible et fournit des services similaires, quoique plus restreints, au Chaos Monkey de Netflix. Cette librairie délivre les fonctionnalités telles que l'ajout de latence, la terminaison de conteneur et la levée d'exception. Le framework SpringBoot utilise le principe de la programmation orientée aspect pour ajouter cette nouvelle fonctionnalité. La librairie expose un ensemble de points terminaux qui peuvent, de manière programmatique notamment, être manipulés.
Chaque développement de microservice doit être modifié, compilé et déployé pour bénéficier de l'injection de fautes.

| ID | Description | Methods |
|---|---|---|
| /chaosmonkey | Running Chaos Monkey configuration | GET |
| /chaosmonkey/status | Is Chaos Monkey enabled or disabled? | GET |
| /chaosmonkey/enable | Enable Chaos Monkey | POST |
| /chaosmonkey/disable | Disable Chaos Monkey | POST |
| /chaosmonkey/watcher | Running Watcher configuration.<br><br>NOTE: Watcher cannot be changed at runtime, they are Spring AOP components that have to be created when the application starts. | GET |
| /chaosmonkey/assaults | Running Assaults configuration | GET |
| /chaosmonkey/assaults | Change Assaults configuration | POST |

FIGURE 5.5 – Chaos Monkey Spring-Boot Wilms (2018)

**Services Cloud**

Pour certains acteurs du *Cloud* il existe des fonctionnalités d'ingénierie du chaos pour leurs services. Microsoft propose le « service-fabric-controlled-chaos » Microsoft (2018), pour Amazon(AWS) artillery.io délivre le « Chaos Lambda » artillery.io (2015), Gremlin Inc. (2016) propose un outil complet qui fonctionne pour de nombreux fournisseurs.

## 5.6 Évaluation des résultats

L'efficacité de l'approche par rétro-ingénierie peut s'évaluer par rapport au système ciblé. En classification de données, en théorie de l'évolution des systèmes d'information, notamment, deux notions sont, entre autres utilisées : le rappel et la précision (recall and precision en anglais).
Pour un univers donné, le rappel mesure le taux d'éléments pertinents trouvés (vrai positif, VP), par rapport au nombre d'éléments pertinents existant (vrai positif, VP + faux négatif FN). Une méthode qui trouve quatre éléments sur cinq présents aura une précision de 80%.

$$Rappel = \frac{VP}{VP+FN}$$

La précision mesure le taux d'éléments pertinents trouvés (vrai positif, VP) par rapport aux éléments trouvés (vrai positif, VP + faux positif FP). Une méthode qui trouve cinq éléments, dont quatre sont pertinents, aura une précision de 80%.

$$Precision = \frac{VP}{VP+FP}$$

Selon le domaine étudié, il est préférable d'accorder plus d'importance à une méthode qui offre un haut taux de rappel, au détriment de la précision, alors que d'autres accordent une importance particulière à la précision, au détriment du taux de rappel.
La F-mesure (F-score, en anglais), combine les deux pour fournir une moyenne. La $F_1$ se calcule comme suit :

$$F_1 = 2 \times \frac{rappel \times precision}{rappel + precision}$$

# Deuxième partie

# Ingénierie du chaos, approche par rétro-ingénierie

**Cette partie présente l'approche par rétro-ingénierie. Elle est divisée en trois chapitres. Le premier est une vue générale. Le suivant décrit la méthode de cette discipline et les différentes étapes du processus. La dernière présente deux études de cas.**

# Chapitre 6

# Rétro-ingénierie : introduction

En utilisant une méthode de rétro ingénierie, est-il possible de réduire voire supprimer l'approche intuitive lors de la sélection judicieuse d'une cible de l'injection de fautes ?
Cette méthode offre-t-elle une aide à la décision qui permet de réduire les connaissances nécessaires à cette sélection ?

La présente partie va tenter de répondre à ces deux questions. La méthode de rétro-ingénierie choisie est constituée de différentes étapes, présentées dans le processus de la figure 6.1.

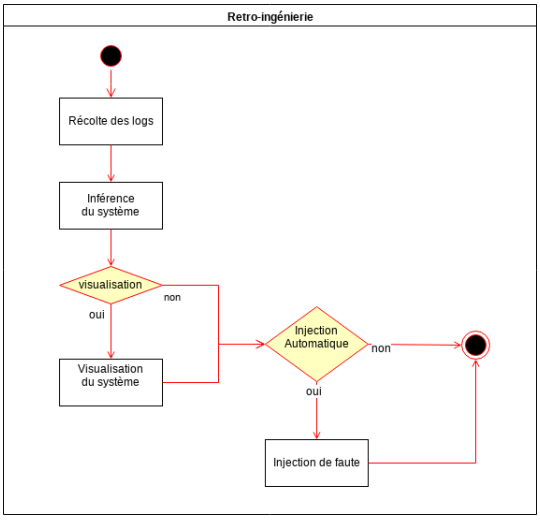


FIGURE 6.1 – Processus de rétro-ingénierie

# Chapitre 7

# Approche par rétro-ingénierie

Le style d'architecture microservices produit des applications largement distribuées tandis que l'organisation des développements s'oriente vers une agilité extrême, avec des modifications du système plusieurs fois par jour. Les ingénieurs travaillant dans de tels systèmes, les qualifient de chaotiques. Une mauvaise configuration peut passer inaperçue dans un premier temps et entraîner par la suite la chute du système ou l'indisponibilité d'une fonctionnalité majeure.
Pour vérifier la robustesse de tels systèmes, une nouvelle discipline a vu le jour : l'ingénierie du chaos. En partant d'un état stable du système, les ingénieurs vont provoquer une injection de fautes. Ils vérifient, avec des métriques métiers, que le système se comporte normalement. Le choix de la cible de l'injection de fautes est intuitif. Il dépend de la connaissance technique et métier du système par l'ingénieur qui la provoque. La plus grande faiblesse de cette approche est le manque de scalabilité qu'entraîne l'utilisation d'un personnel très qualifié et connaissant en profondeur le système.

Peut-on améliorer la robustesse d'un système hautement distribué, sans en posséder une connaissance détaillée, en utilisant la rétro-ingénierie ? Est-il possible d'appliquer la rétro-ingénierie pour analyser l'évolution de l'organisation d'un système d'information complexe, hautement distribué ?
Ce mémoire propose une méthode de rétro-ingénierie pour un système hautement distribué. Cette méthode doit permettre d'apporter une information suffisante pour limiter l'approche intuitive de l'ingénierie du chaos, sans nécessiter une connaissance préalable du système.
La suite de ce chapitre exposera la méthode de rétro-ingénierie. Pour commencer, elle spécifiera le champ d'application. Un système distribué peut être implémenté de bien des manières. Cette méthode s'adressera aux systèmes utilisant le style REST dans une infrastructure basée sur des conteneurs

et des microservices. Ensuite, elle s'attardera à la récolte des informations nécessaires. Pour terminer elle détaillera l'exploitation de l'information : l'exploration du système, sa visualisation, la découverte de points de faiblesses potentielles et l'injection de fautes.

## 7.1 Contexte d'applicabilité

Avant d'explorer plus avant la rétro ingénierie d'un système chaotique, il est nécessaire de préciser le contexte de cette étude. Si la majorité des infrastructures disponibles dans le Cloud pour réaliser une architecture microservices utilise un style d'architecture REST, d'autres approches sont possibles. Dans l'évolution de systèmes d'information, des éléments tels qu'un ESB, un mainframe ou un protocole de message asynchrone propriétaire peuvent être imbriqués avec une architecture plus récente utilisant des microservice. Le contexte de cette recherche va se restreindre à l'environnement initial de l'ingénierie du chaos : un système hautement distribué utilisant le style d'architecture REST, des microservices, une infrastructure utilisant des conteneurs et le protocole de communication HTTP(S).
Le style d'architecture REST est particulièrement adapté à la découverte du système, grâce à l'identification unique des ressources.

## 7.2 Méthode de rétro-ingénierie

Un système basé sur une architecture hautement distribué est un système réparti entre différents endpoints. On peut représenter un tel système sous la forme d'un graphe orienté, où un nœud représente un endpoint et un arc représente une requête.
Le but de la rétro-ingénierie est de découvrir le graphe orienté de ce système.

## 7.3 Récolte d'informations

### 7.3.1 Collecter l'information

#### Logs centralisés

De nombreux composants produisent des informations sous la forme de fichiers log. Dans une architecture largement distribuée, c'est pratiquement un prérequis de les agréger dans un système de logs centralisés. Un tel système possède des fonctionnalités permettant l'exploitation des informations et l'exportation sous différents formats.

#### Informations disponibles

Les informations disponibles dans les logs, concernant les requêtes, sont nombreuses et diverses. Mais on y retrouve souvent les informations nécessaires à la rétro-ingénierie du chaos : l'URI de la cible, le verbe REST, un *timestamp* et un identifiant de corrélation. Si celui-ci n'est pas présent, il peut être injecté par différents composants dans l'entête de la requête. Des composants utilisés couramment comme *façade*, tel que les serveurs *Apache*, *N-ginx* ou les API gateways, possèdent de telles fonctionnalités de base.

## 7.4 Découverte du système

### 7.4.1 Rétro-ingénierie sur les logs

À partir des données déposées dans le système de centralisation des logs applicatifs, il est possible d'appliquer un premier processus de rétro-ingénierie pour tenter de découvrir le système. Le processus est le suivant : (1) pour chaque URI distinct d'une requête, créer une représentation d'un endpoint ;(2) en utilisant l'identification de corrélation, créer une représentation d'un lien entre deux endpoints ;(3) l'ensemble des relations d'une même identification de corrélation représente une fonctionnalité du système.

#### Noeud (*endpoint*)

Une ligne d'un log applicatif peut se résumer comme la suite des éléments suivants : timestamp, autorité, chemin, verbe, paramètres de la requête, code HTTP de la réponse, l'ID de corrélation, etc. La concaténation de l'autorité, du chemin et du verbe représente l'identifiant de l'endpoint.

Listing 7.1 – nginx log

```
[29/Oct/2018:11:23:03 +0000] - 172.17.0.2 - "GET
    api/number.html HTTP/1.1" 200 "Id: f66dd2c5f1c..."
```

Dans l'exemple ci-dessus, l'identifiant de l'endpoint est
« GET 172.17.0.2/number.html ».
La norme RFC 3986 n'inclut pas l'utilisation du verbe HTTP, car cette spécification supporte des protocoles qui n'en ont pas l'usage. Néanmoins, pour les besoins de la rétro-ingénierie, quand le protocole HTTP est utilisé, le verbe fera partie de l'identification de l'endpoint.
Tous les endpoints utilisés lors de l'enregistrement des journaux sont découverts.

#### Arc(requête)

Grâce à la présence d'identifications de corrélation et de l'ensemble des endpoints, il est possible de découvrir l'origine d'une requête, notamment

en utilisant la chronologie des enregistrements. L'origine et la cible de la requête donnent naissance à une relation.
En outre, l'identification de corrélation identifie un ensemble de relations qui forment une demande unique. Celle-ci permet de découvrir une fonction du domaine du système.
Toutes les fonctions utilisées lors de l'enregistrement des journaux sont découvertes.

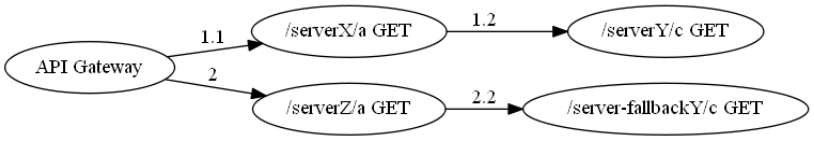


FIGURE 7.1 – Découverte du système

### Résultat provisoire

Si la rétro-ingénierie permet de découvrir toutes les fonctions utilisées du système, lors de l'enregistrement des logs, elle ne garantit nullement que l'ensemble des relations entre les points terminaux soit révélé. Elle ne permet pas de déduire que deux endpoints sont des instances différentes d'une même fonction ni d'en conclure une quelconque redondance.

### 7.4.2 Approche bayésienne

Lors de l'analyse des logs, chaque ligne n'indique probablement pas une nouvelle relation distincte. Deux relations désignées par la même origine et la même cible représentent par définition deux instances d'une même relation. Il est trivial de les identifier et d'éliminer les doublons. Il est moins aisé cependant de deviner, par leurs similitudes, que deux relations effectuent la même opération.
En considérant toute la durée de vie d'un système, sans modification de celui-ci, l'ensemble des logs va représenter la totalité des relations constituant celui-ci. En y pratiquant des inférences statistiques, il sera alors possible d'indiquer que telle relation est similaire à une autre. Mais dans le cadre de l'ingénierie du chaos, il est crucial de découvrir le fonctionnement du système, avant la fin de celui-ci. La découverte du système par rétro-ingénierie utilisera des données qui seront récoltées pendant une période restreinte.

#### Rétro-ingénierie bayésienne

Effectuer une inférence bayésienne, c'est formuler une hypothèse pour expliquer un ensemble de données. Appliquer à la rétro-ingénierie d'un système distribué, dans le cadre de l'ingénierie du chaos, c'est formuler une hypothèse à propos de (1) $n$ composants sont sémantiquement identiques et (2) $m$ suites de composants sont des fonctionnalités du système, à sémantique identique.

En langage formel :

$f \in F$ — soit $f$ un service *Rest* appartenant au système $\mathbb{F}$ , (7.1)

$f\prime() \equiv f() \iff \forall x, y | \{y = f(x) \wedge y' = f'(x)\} \Rightarrow y = y'$ — soit $f\prime$ est équivalent à $f$ , (7.2)

$\varphi(f()) \stackrel{\text{def}}{=} \varphi \in \mathbb{F} \wedge \varphi \xrightarrow{call} f()$ — soit $\varphi(f())$ un client de $f$ , (7.3)

$\varphi\prime(f()) \equiv \varphi(f()) \iff \varphi\prime \xrightarrow{call} f() \vee \varphi\prime \xrightarrow{call} f\prime()$ — soit $\varphi\prime(f())$ est équivalent à $\varphi(f())$ (7.4)

$Sv = (f()_n) | f()_{n-1} = \varphi(f()_n)$ — soit $Sv$ une fonctionnalité, une suite de services *Rest* , (7.5)

$Sv\prime \equiv Sv \iff (f()_n) | f()_{n-1} = \varphi(f()_n \vee \varphi\prime(f()_n)$ — soit $Sv\prime$ est équivalent à $Sv$, (7.6)

$\Omega(Sv, m) \stackrel{\text{def}}{=} |f()_{n_i}| > m \wedge |\varphi(f()_{n_i})| > m$ — soit $\Omega$ une fonctionnalité robuste à $m$ pannes, par définition possède : $m$ services équivalents et $m$ clients équivalents pour chaque service de la suite. (7.7)

L'approche de Bayes *pure* consiste à découvrir $Sv\prime \equiv Sv$, donc chaque $f\prime \equiv f$ et $\varphi\prime(f) \equiv \varphi(f)$. En considérant les données récoltées par la rétro-ingénierie :

— $f$ est identifié par un URI ;
— $\varphi$ est identifié par un URI qui a une relation vers $f()$ ;
— $Sv$ est identifié par tous les URI qui partagent le même identifiant de corrélation.

L'inférence bayésienne est alors :

$$P[f \equiv f' | data] = \frac{P[data | f \equiv f'] P[f \equiv f']}{P[data | f \equiv f'] P[f \equiv f'] + \sum_{alter} P[data | alter] P[alter]}$$

*alter* pour les données alternatives.

#### Complexité algorithmique

La complexité temporelle est de $\Theta(n^2)$ soit un tableau à deux dimensions. Appliquée aux données, cela signifie calculer chaque ligne contenant un URI.
En considérant un temps de lecture minimum théorique pour un disque dur de 500 Mo par seconde, il faut 390 nanosecondes pour lire une ligne de 0,2Ko. Le temps de calcul minimum est de 152 secondes pour 1.000 lignes et de 422,5 heures pour 100 000 lignes.

### 7.4.3 Approximation de Bayes appliquée

Ce mémoire n'a pas pour ambition de faire une recherche exhaustive dans le domaine de l'intelligence artificielle, ni d'implémenter un algorithme d'apprentissage. Pour une première approche, la rétro-ingénierie du chaos utilisera une approximation de Bayes qui ignorera la fonction de partition.

$$P[f \equiv f' | data] = \frac{P[data | f \equiv f'] P[f \equiv f']}{P[data]}$$

#### Modèle à forte croyance

Les modèles d'inférence pour identifier deux services équivalents ne sont heureusement pas aussi complexes que ceux utilisés dans la classification d'images. Un microservice est une unité exécutable de déploiement qui, lors de son utilisation dans un système, sera instancié plusieurs fois dans des conteneurs différents. Les URI des endpoints sont alors identiques, à l'exception de leur autorité.
Selon les pratiques de l'organisation, il peut être en revanche hasardeux d'utiliser le *path* pour inférer sur une quelconque équivalence. Un *path* de cinquante caractères contenant « v1 » pour indiquer la version première du service a une distance de Levenshtein normalisée de 0,02 (0 indiquant totalement identique et 1 totalement différent) par rapport au *path* qui change la version en « v4 ». Cette faible distance ne reflète pas la grande différence de fonctionnalité entre les deux services. À contrario, «/cust » et «/distribuedMemoryCache-cust » peuvent être des *paths* de services équivalents alors qu'ils ont une distance de Levenshtein normalisée de 0,85.

Le choix des modèles d'inférence dépend donc du système à découvrir. Une simple comparaison de l'adresse de l'endpoint peut suffire pour une majorité de cas, tandis qu'un modèle plus poussé est nécessaire pour les autres.

Après avoir récolté les informations à propos du système et découvert celui-ci en pratiquant des inférences bayésiennes, il reste à exploiter les résultats. La section suivante présente leur visualisation.

## 7.5 Visualisation des résultats

La rétro-ingénierie du chaos s'appui fortement sur la visualisation. Si elle a pour ambition de s'affranchir, en partie, de l'intervention d'ingénieurs qualifiés, elle a toujours pour mission de fournir une meilleure compréhension du système. En utilisant une approche statistique pour découvrir celui-ci, l'ingénierie du chaos ne peut offrir une preuve de la robustesse du système, au contraire du LDFI. L'approche bayésienne fera un pari sur l'état du système, selon les crédences observées. Il restera alors à l'ingénieur du chaos la lourde responsabilité d'entériner ou pas celui-ci. Pour l'aider dans cette tâche la rétro-ingénierie doit offrir une visualisation du système et des aides à la décision.

### 7.5.1 Visualiser le système

Selon le niveau d'abstraction choisi, un système devient relativement simple à comprendre, au prix d'une perte substantielle d'informations. La visualisation gagnera en efficacité si elle propose un mécanisme permettant de changer le niveau d'abstraction.
Pour l'étude de l'évolution d'un système, la dimension temporelle est particulièrement pertinente. La visualisation devra permettre un parcours aisé à travers les changements dans le temps.
Enfin, le choix entre une vision statique ou dynamique apportera un avantage particulier. Si un manque de redondance gagne à être découvert sur une représentation figée, une panne cyclique sera plutôt mise en évidence sur une représentation par flux.

### 7.5.2 Décoration

Parmi toutes les informations récoltées – notamment leurs corrélations, leurs inférences et les différents calculs réalisés - nombreuses sont celles qui présentent un intérêt certain dans l'aide à la décision. On peut par exemple citer le volume de données en transit par endpoint, le pourcentage d'utilisation des chemins, ou le degré d'inférence entre différents endpoints.
Le système de visualisation les représentera sous forme d'une décoration particulière, d'une mise en évidence par couleur ou affichera simplement, à la demande, les données.

La visualisation apporte une aide à la compréhension du système et au choix d'une cible pour l'injection de faute. Mais est-elle nécessaire ? La section suivante tentera de répondre à cette question.

## 7.6 Injection de fautes

La rétro-ingénierie du chaos peut détecter des faiblesses dans un système largement distribué, ou plutôt, elle peut en faire le pari -tout comme le médecin de famille qui face aux symptômes, déclare qu'une femme est enceinte, sous réserve des résultats de la prise de sang. Une inférence bayésienne reste un raisonnement où subsiste un certain niveau d'incertitude. Incertitude qui pourrait être levée par une injection de fautes. Soit le système est redondant et la rétro-ingénierie découvrira une nouvelle relation ou augmentera le crédit à apporter à l'équivalence de deux services, soit-il ne l'est pas et, à condition que cela soit détecté, le doute sera levé sur la faiblesse avérée du système.
Il est cependant possible de tirer parti de l'avantage qu'offre la rétro-ingénierie du chaos en termes de moindre ingérence. Dans certains domaines, comme l'assistance médicale, la commande d'automates, la bourse, etc., il est difficile d'admettre l'injection de fautes qui pourraient avoir de lourdes répercussions. Dans les contextes où l'injection de fautes en production n'est pas une option retenue, la rétro-ingénierie du chaos propose une alternative. Bien en retrait du système de production, elle peut analyser les informations dans les journaux de celui-ci, calculer des inférences bayésiennes, concevoir une visualisation et enfin offrir une aide à la décision aux ingénieurs en charge du système. Un défaut dans la robustesse du système pourra être transformé en risque et géré en tant que tel par l'entreprise.
Néanmoins, l'injection de fautes reste irremplaçable dans un système largement distribué pour autant qu'on en accepte les conséquences éventuelles. Une inférence bayésienne aura bien du mal à mettre en avant une faille dans un système qui semble robuste. Comment pourrait-elle deviner que la perte d'une seule instance d'un service redondant entraînera la chute du système, après un temps qui semble aléatoire ? Selon les données analysées, notamment les temps de réponse, une inférence bayésienne plus avancée pourrait mettre en avant ce genre de fragilité, mais probablement en conjonction avec l'injection de fautes.

Que l'on choisisse de réaliser, ou non, une injection de fautes, dans le domaine de la rétro-ingénierie, il est important de valider les résultats obtenus et de comparer ce qui a été découvert avec le système réel. La section suivante présente l'évaluation des résultats.

## 7.7 Évaluation des résultats

L'approche par rétro-ingénierie du chaos sera appréciée selon les critères de rappel et de précision à l'aide d'une F-mesure (F-score, en anglais). Le rappel et la précision ayant autant d'importance pour découvrir des failles potentielles dans la robustesse du système.
Le score $F_1$ sera appliqué pour la découverte des fonctionnalités du système et pour les points de fragilité.
Une nuance doit cependant être apportée à cette notion de fragilité. La rétro-ingénierie du chaos détecte un potentiel manque de redondance d'un endpoint pour une fonctionnalité donnée, ou permet de visualiser un comportement particulier qui demande l'analyse d'un ingénieur. Elle est incapable d'anticiper le manque de robustesse interne d'un point de terminaison. Or, le manque de robustesse interne peut entrainer la chute du système.

Lorsque la rétro-ingénierie est appliquée à la découverte des mécanismes internes d'une fonctionnalité, dans le but par exemple de la reproduire, la réussite de l'approche peut se mesurer par rapport aux spécifications de cette fonctionnalité. Dans le domaine de l'ingénierie du chaos, le système étudié est non seulement inconnu, du moins partiellement, mais surtout soumis à d'incessantes variations. Dans ce contexte, l'évaluation des résultats peut être difficile à réaliser.

Cette section termine le chapitre présentant l'ingénierie du chaos par une approche par rétro-ingénierie. Celle-ci doit maintenant être éprouvée. Le chapitre suivant présentera deux études de cas. La première constitue le baptême du feu de l'approche, dans un laboratoire dédié, tandis que la seconde s'exposera aux difficultés du terrain d'un système de production.

# Chapitre 8

# Étude de cas

Ce chapitre présente deux études de cas de rétro-ingénierie. La première aborde l'application de la méthode dans un environnement aseptisé et contrôlé d'un laboratoire. La seconde présente une variante réalisée sur un système de production.

## 8.1 Étude de cas 1 : laboratoire de rétro-ingénierie

Le système idéal pour valider l'approche par rétro ingénierie serait un système hautement distribué basé sur une architecture REST utilisant des microservices. Chaque composant produirait des logs applicatifs normalisés qui seraient centralisés dans un système dédié. Mais le plus important est qu'il serait documenté ou du moins relativement maîtrisé, car sans une bonne connaissance du système, il serait difficile de vérifier les résultats obtenus.
Netflix possède sans doute un tel système, comme d'autres géants d'internet. Malheureusement, leurs logs sont tellement volumineux que le système de logs est basé sur un flux avec analyse en continu. Il faudrait donc se greffer sur ce système, ce qui dépasse le cadre du mémoire.
La récupération des logs applicatifs d'un large système et l'obtention de sa documentation, provenant d'une entreprise soulèvent des problèmes juridiques, de confidentialité, de volumétrie et de pertinence.

La conception d'un laboratoire permet de s'affranchir de nombreuses contraintes liées aux systèmes de production en perpétuelle évolution. Il permet également de produire une documentation exacte et détaillée du système étudié, afin de valider l'approche par rétro-ingénierie.

Le laboratoire choisi n'est pas la reproduction d'une application distribuée avec toute son infrastructure coûteuse, mais il produit les mêmes éléments nécessaires à la rétro-ingénierie : les logs.

Ce laboratoire se compose d'un générateur de fichiers logs simulant un système largement distribué. Il permet de créer des systèmes virtuels se composant d'un certain nombre de services, composés eux-mêmes d'un certain nombre de endpoints. Le système généré peut être résistant à plusieurs pannes ou comporter des faiblesses.

Le laboratoire fonctionne en deux étapes. La première consiste à créer un système virtuel tandis que la seconde génère un log centralisé, simulant son exécution.
Le système généré peut enregistrer les interactions virtuelles dans un fichier. Sur un poste de travail récent, il produit en cinq secondes, avec deux *threads*, 440 000 lignes de logs.

Enfin, le laboratoire fournit la documentation du système virtuel sous forme d'une représentation visuelle afin de faciliter sa compréhension.

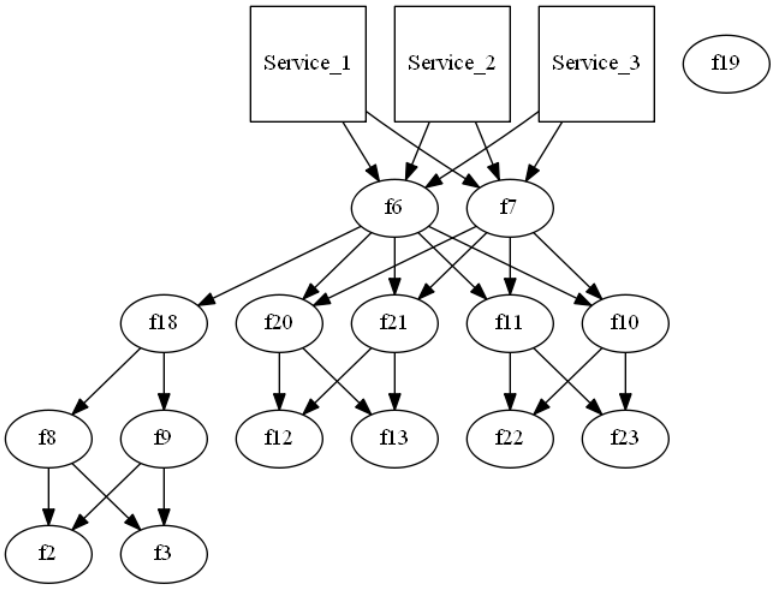


FIGURE 8.1 – Documentation du système généré par le laboratoire

### Pertinence du laboratoire

L'utilisation d'un laboratoire, produisant des logs dans la validation de l'approche par rétro-ingénierie du chaos offre plusieurs avantages. Pour commencer, par définition un laboratoire est un environnement isolé des événements inattendus qui se produisent dans un véritable système de production. Ensuite, la récolte, la normalisation et le filtrage des informations par les logs peuvent être des tâches très consommatrices de ressources. Pour termi-

ner, la mise en place d'un véritable système hautement distribué représente un coût non négligeable en matériel ou en location dans le Cloud.

### 8.1.1 Méthode de rétro-ingénierie

La méthode appliquée pour cette première rétro-ingénierie du chaos est une fidèle application de l'approche détaillée dans le chapitre précédent. Pour commencer, les informations sont récoltées et centralisées. Ensuite le système est découvert par des inférences bayésiennes et une représentation visuelle est créée. Pour terminer, la rétro-ingénierie du chaos est évaluée selon les critères préétablis.

### 8.1.2 Récolte d'informations

La récolte et la centralisation des informations sont grandement facilitées par la production d'un fichier uniformisé par le laboratoire. Ce fichier est composé de lignes dont la structure est la suivante :

Id , timestamp , serveur , verbe , path , ID de correlation

L'identifiant en tête de ligne désigne un service REST. Il n'est pas utilisé par la rétro-ingénierie, mais peut être utile pour investiguer des erreurs.
La différence notable entre le format de cette ligne et le format d'une ligne d'un log d'un serveur NGINX, utilisé par exemple en façade d'un service Rest, est la présence de nombreuses données inexploitées par la rétro-ingénierie du chaos : informations sur le client (navigateur et système d'exploitation, entre autres), adresse du serveur, etc.

L'exploitation du fichier de journalisation consiste à utiliser une structure de données adaptée, en utilisant l'identifiant de corrélation pour découvrir les fonctionnalités.

La rétro-ingénierie dispose à cette étape intermédiaire d'un ensemble d'endpoints, et d'un ensemble de fonctionnalités.
Pour un système redondant, il reste à identifier les endpoints et les fonctionnalités qui sont considérés comme équivalents fonctionnellement.
À ce stade de la méthode, pour un système donné, soit 24 endpoints et 3 services, la rétro-ingénierie du chaos a trouvé 15 endpoints et 24 services. Soit un score médiocre de :

$$\begin{aligned} Rappel : 1 &= \frac{3}{3} \\ Precision : 0,125 &= \frac{3}{3+21} \\ F_1 : 0,22 &= 2 \times \frac{1 \times 0,125}{1+0,125} \end{aligned}$$

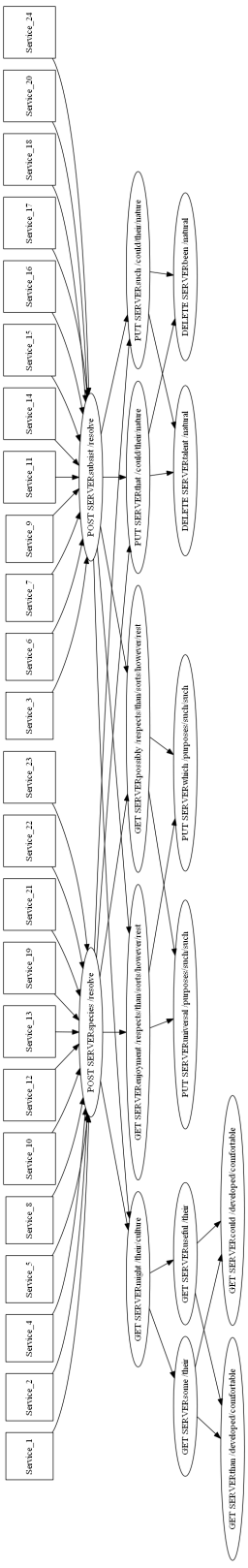


FIGURE 8.2 – Étape intermédiaire : le système brut découvert

### 8.1.3 Induire le système par inférence de Bayes

À partir des structures de données réalisées aux étapes précédentes la découverte du système se poursuit en réalisant des inférences bayésiennes, sur les endpoints, et ensuite sur les services.

1. Pour la similarité de deux endpoints, la croyance peut s'établir sur différents critères. Le plus simple est sans doute qu'ils partagent le même URI, à l'exception de leur autorité (le nom ou l'adresse du serveur). Selon les pratiques de l'organisation qui a conçu le système, le calcul d'un score basé sur la distance de Levenshtein pourrait être utilisé.
   Pour cette étude de cas, les deux critères ont été retenus. La similitude des endpoints se calcule comme ceci :

$$P[f \equiv f'|data] = \frac{P[data|f = f']P[f = f']}{P[data]}$$

$P[data|f = f'] = 1$ ou moins selon la qualité des données

$P[f \equiv f'] = 1$ si seul le serveur est différent, 0 sinon

$P[data] = 1$ simplification de l'inférence de Bayes

$\rightarrow P[f \equiv f'|data] = \{0|1\}$

2. Pour la similarité des services, le critère basé sur la distance de Levenshtein établi une probabilité comme suit :

$$P[Sv \equiv Sv'|data] = \frac{P[data|Sv = Sv']P[Sv = Sv']}{P[data]}$$

$P[data|Sv = Sv'] = 1$ ou moins selon la qualité des données

$P[Sv \equiv Sv'] = P[f_1 \equiv f'_1] + ... + P[f_n \equiv f'_n]$ équivalent la selon distance de Levenshtein

$P[data] = n$ le nombre de points de terminaison

$$\rightarrow P[Sv \equiv Sv'|data] = \frac{P[f_1 \equiv f'_1] + ... + P[f_n \equiv f'_n]}{n}$$

Après les inférences bayésiennes le système donné est composé de trois services. L'évaluation de la méthode est de :

$$Rappel : 1 = \frac{3}{3}$$
$$Precision : 1 = \frac{3}{3}$$
$$F_1 : 2 = 2 \times \frac{1 \times 1}{1+1}$$

Soit le score $F_1$ maximal.

### 8.1.4 Visualisation des résultats

La production de la représentation visuelle est réalisée avec Graphviz. Le graphique ci-dessous est généré à partir d'un fichier DOT qui est écrit par l'application.

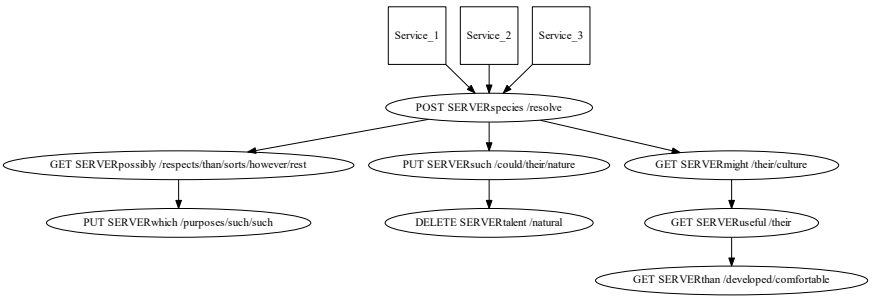


FIGURE 8.3 – Fonctionnalités découvertes

Cette représentation peut sans aucun doute être enrichie. Il serait intéressant de connaître la redondance pour chaque point de terminaison, de visualiser le nombre de requêtes pour chaque arc. Dans ce cas, « une technique courante utilisée dans les pages Web pour créer un contenu interactif basé sur des graphes, est de s'appuyer sur un serveur *webdot* HTTP. Il est invoqué par un URL qui spécifie un fichier graphique distant à récupérer, la mise en page Graphviz à exécuter et le type MIME de l'image à créer. [...] En outre pour fournir des images en ligne, si un nœud ou une arête du graphique spécifie un attribut URL, l'image correspondante agira comme un lien vers cette URL. » Ellson et al. (2003)

### 8.1.5 Découverte de failles

Le système est maintenant découvert, mais il manque toujours une information primordiale. Ce système hautement distribué possède-t-il des points de fragilité ? Pour le découvrir, l'algorithme du LDFI est utilisé. Une fonctionnalité est robuste à $n$ pannes si chaque point de terminaison qui la compose est appelé par $n + 1$ points de terminaison.
L'implémentation est la suivante :

```
Pour chaque service
    Pour chaque point de terminaison
        le point de terminaison possede n+1
          autres points de terminaison similaire
        Pour tous les points de terminaison similaire
            il existe n+1 appels depuis
              un point de terminaison similaire
               au point de terminaison precedent de ce service.
```

L'algorithme détecte dans notre système de laboratoire une fonctionnalité qui possède au moins un point de faiblesse. La représentation visuelle est la suivante :

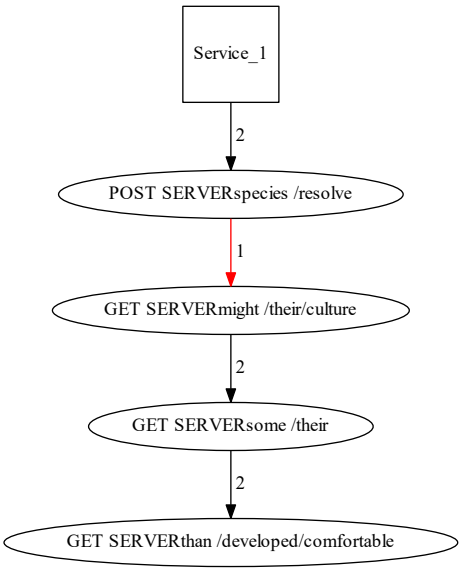


FIGURE 8.4 – Service fragile découvert

Avec un système laboratoire qui simule son fonctionnement en produisant un fichier de journalisation, il n'est pas possible de produire une injection de faute pour démontrer la faiblesse du système. Cependant, une des prémisses de l'ingénierie du chaos est la croyance que le système résistera bien à l'injection de fautes, sans impact significatif sur la disponibilité de ses fonctionnalités. Or dans notre cas, il est trivial de constater qu'une injection de fautes sur le point de terminaison non redondant provoquera une panne dans le système, à moins d'avoir la connaissance qu'un autre point de terminaison non découvert lui soit similaire.

### 8.1.6 Évaluation des résultats

L'ingénierie du chaos, par méthode de rétro-ingénierie, appliquée sur un système aseptisé d'un laboratoire a donné les résultats suivants : toutes les fonctionnalités du système ont été découvertes, sans aucun faux positif ; la fonctionnalité possédant un point de fragilité a été découverte, sans aucun faux positif ; seuls les points de terminaison utilisés par les fonctionnalités ont été découverts, soit 62 %.
La méthode étant basée sur l'analyse de logs, il est donc impossible à l'ingénierie du chaos de découvrir les endpoints qui ne sont pas mentionnés dans ceux-ci.

### 8.1.7 Conclusion de l'étude de cas

L'étude de cas concerne un système produit par un laboratoire. Elle a été réalisée avec une implémentation simple et naïve de l'ingénierie du chaos. Les résultats donnent un score maximum, ce qui était attendu d'une rétro-ingénierie appliquée sur un système non-soumis aux aléas d'un véritable système de production. Pour des fichiers logs dont la qualité serait moindre, l'utilisation d'une approche bayésienne apporte une plus grande souplesse à la détection de similarités.

L'approche par rétro-ingénierie a permis de découvrir l'entièreté du système utilisé ainsi qu'un point de faiblesse potentiel. Le système étudié étant généré aléatoirement, l'intuition est complètement absente du processus de décision.

## 8.2 Étude de cas 2 : cartographie d'une architecture SOA

Cette étude de cas a été réalisée dans le cadre d'un projet de cartographie pour un système basé sur une architecture SOA d'une administration régionale. Par le passé, les tentatives pour réaliser une cartographie des applications et leurs interactions ont échoué car le temps de réalisation était trop long et la maintenance de la cartographie trop onéreuse. Une nouvelle approche a alors été décidée : la création d'une cartographie « automatique » s'appuyant sur des logs de production.
Il ne s'agit donc pas, à proprement parler, d'ingénierie du chaos. Néanmoins, une des parties prenantes de l'initiative désire que la cartographie puisse offrir une aide à la décision lors des analyses d'impact pour des changements sur les composants d'infrastructure.

### 8.2.1 Connaissance préalable du système

Les personnes du métier n'ont aucune connaissance du flux d'information. L'équipe « process métier » ne participe pratiquement pas à la conception des applications.
Parmi le personnel de la Direction des Systèmes d'Information (DSI), la connaissance est fragmentée.

- Un architecte maîtrise la logique des flux d'information principaux.
- Les ingénieurs système maîtrisent l'infrastructure. Il y a une documentation, et une surveillance permet d'indiquer la bonne santé des applications.
- Les développeurs et architectes d'applications sont des consultants avec un haut taux de remplacement. La documentation est absente ou obsolète. L'analyse du code permet, en partie, de comprendre les liaisons entre les applications.

La robustesse de l'ensemble est garantie par plusieurs règles strictement suivies :

1. Les composants back-end sont sans état (*stateless*) ;
2. Les composants back-end sont toujours installés sur deux serveurs séparés ;
3. Pour faire un appel sur un composant back-end, la seule option est d'adresser la requête à un ESB, qui assure la répartition vers les différents serveurs ;
4. L'ESB est une installation à « haute disponibilité ».

### 8.2.2 Méthode de rétro-ingénierie

La méthode appliquée pour cette rétro-ingénierie est une fidèle application de l'approche détaillée dans la partie précédente. Pour commencer, les informations sont récoltées et centralisées. Ensuite le système est découvert par des inférences bayésiennes, et une représentation visuelle est créée. Pour terminer, la rétro-ingénierie est évaluée selon les critères préétablis.

### 8.2.3 Récolte d'informations

La source des informations est différente pour cette étude de cas. Si le système génère des logs, ils ne sont ni centralisés ni corrélés. Chaque application produit son log comme si elle était autonome.
En revanche, l'ESB est capable de fournir un log de son utilisation au format suivant :

```
Client, cible 1, cible 2, ..., cible n,
```

*cible étant l'URL sans le verbe.*

L'exploitation du log consiste à utiliser une structure de données adaptée, en utilisant le *client* pour découvrir les fonctionnalités.

La rétro-ingénierie dispose à cette étape intermédiaire d'un ensemble d'endpoints, et d'un ensemble de fonctionnalités.
Pour un système redondant, il reste à identifier les endpoints qui sont considérés comme équivalents fonctionnellement.

### 8.2.4 Découverte du système

À partir des structures de données réalisées à l'étape précédente, la découverte du système se poursuit en réalisant des inférences sur les endpoints. Celles-ci sont simplifiées par la topologie et les conventions du système. Deux endpoints sont identiques fonctionnellement si, seul le serveur est différent.

### 8.2.5 Visualisation des résultats

La visualisation est ensuite réalisée avec l'outil *Vizceral*, initialement utilisé par Netflix pour la représentation de son système. Son affichage dynamique permet différentes profondeurs de détail (régions, applications, services, etc.) ainsi qu'une représentation des volumes de transfert (inexploités dans le cas présent).
L'initiative de la cartographie « automatique » se poursuit par manipulation de la présentation.

FIGURE 8.5 – Système découvert, avec indication de points de fragilité

### 8.2.6 Découverte de failles

Pour les besoins de ce mémoire, la découverte de points de faiblesse a été réalisée sur la structure de données et la visualisation a été enrichie. Celle-ci affiche plusieurs points de faiblesse.

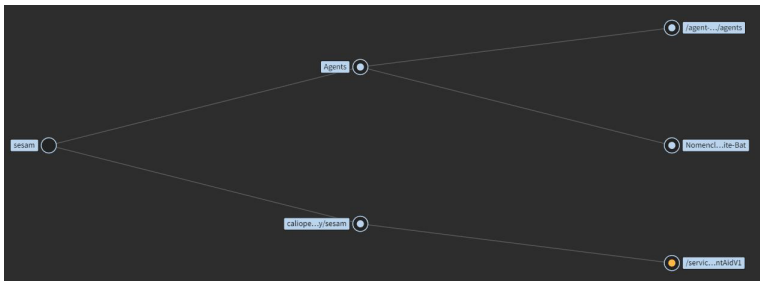


FIGURE 8.6 – Détails d'un service avec une fragilité potentielle

### 8.2.7 Évaluation des résultats

L'évaluation des résultats est délicate car personne ne détient la connaissance complète du système.
La méthode est basée sur l'analyse d'un agrégé d'informations provenant de l'ESB. L'architecte ayant la connaissance des flux a pris le temps d'analyser la pertinence de la visualisation produite. Il n'a pas découvert d'endpoint manquant. Le *recall* est estimé à plus de 90%. A priori, il n'y a aucun faux positif (précision de 100 %).
Certains points de fragilité sont connus, mais non documentés. D'autres demandent une investigation plus approfondie. Pour au moins un cas, une injection de fautes lèverait le doutesur la fragilité d'un point.

### 8.2.8 Conclusion de l'étude d'un système en production

L'étude de cas concerne un système en production. Elle a été réalisée avec une implémentation simple et naïve de l'ingénierie du chaos. L'outil de visualisation apporte un support pertinent lors de l'étude du système.

L'approche par rétro-ingénierie a, probablement, permis de découvrir l'entièreté du système utilisé ainsi que des points de faiblesse non documentés. La visualisation permet de réduire de manière significative les connaissances préalables à propos du système. Elle sera utilisée par les architectes et les développeurs comme une aide à la compréhension de celui-ci.

Ces études de cas terminent la deuxième partie de ce mémoire. Celle-ci a présenté en détail l'ingénierie du chaos par une approche de rétro-ingénierie. Elle est constituée d'une récolte d'informations dans les logs, puis d'une découverte du système par inférence bayésienne pour terminer par une visualisation de celui-ci, avec en option une injection automatique de fautes.

La partie suivante présente les limites et perspectives de cette approche.

# Troisième partie

# Discussion, perspectives

**Cette partie aborde les limites de la rétro-ingénierie dans l'ingénierie du chaos, ainsi que ses perspectives. Elle présente également les conclusions du mémoire.**

# Chapitre 9

# Critique de l'approche

Ce chapitre présente les limitations de l'approche par rétro-ingénierie, en commençant par les limites intrinsèques de la méthode, pour poursuivre avec les limites extrinsèques. Il se termine en revisitant les limitations de l'ingénierie du chaos.

## 9.1 Limites intrinsèques de la méthode de rétro-ingénierie

### Utilisation du système

Une fonctionnalité ou un endpoint qui ne sont pas utilisés pendant la période d'analyse des logs seront invisibles pour la rétro-ingénierie. Il peut s'agir d'un composant d'une importance capitale mais qui est sollicité quelques fois par mois.
D'autres types de composants sont, étant donné leur nature, faiblement utilisés dans un usage nominal du système. On peut citer ceux qui s'activent uniquement en cas de panne ou de congestion, ou ceux qui s'utilisent lors de la récupération du système après un désastre, ou encore ceux qui réagissent en cas d'intrusion. Ils seront probablement absents des résultats, provoquant dans certains cas des faux positifs dans l'analyse de la robustesse.

### Extrême variabilité du système

L'architecture microservices se base sur des composants sans état. Un container est conçu pour être évincé et redémarré selon les besoins de l'orchestrateur. Si l'on ajoute les changements journaliers sur la configuration du système, cela produit un système à grande variabilité. La méthode de rétro-ingénierie pourrait confondre les composants détruits avec ceux qui ne sont pas encore ou plus utilisés.
La visualisation représenterait un système qui n'existe plus et il faudrait

faire appel, à nouveau, à l'intuition de l'ingénieur pour en donner une interprétation correcte.

### 9.1.1 Robustesse par redondance

La méthode se base sur le postulat qu'une fonctionnalité est robuste à la panne de $n$ nœuds, si chaque composant est répliqué à $n + 1$ nœuds. Ce postulat présuppose que leur nombre restant, après la perte de $n$ instances, est capable de supporter la charge, sans provoquer de panne sur la fonctionnalité. Or, sur certaines configurations, la perte de composants entraîne des effets de bord imprévus, menaçant la stabilité du système. Sur de telles configurations, l'injection de fautes est particulièrement riche en enseignement. Malheureusement, la rétro-ingénierie ne fournit pas la moindre information à ce sujet. La visualisation alliée à une faible connaissance du système conduira probablement à un faux sentiment de sécurité.

Cette limite est le point majeur de divergence entre une approche par rétro-ingénierie et une approche par intuition. L'approche par intuition n'est pas une conséquence de la méthodologie utilisée par les ingénieurs du chaos : c'est un choix réfléchi et justifié. Il en est de même pour la connaissance préalable du système qui, elle aussi, est délibérée.

## 9.2 Limites extrinsèques de la méthode de rétro-ingénierie

**Présence de logs exploitables**

La méthode de rétro-ingénierie choisie est basée sur l'exploitation des logs de production. Toute partie du système qui ne produit pas de logs est, de facto, invisible. S'il reste possible d'instrumentaliser un composant central, tel qu'une passerelle API (API Gateway), cela fera apparaître uniquement les éléments qui lui sont attachés.

La présence de logs n'est pas une condition suffisante. Encore faut-il qu'ils soient exploitables. Certains composants suppriment les entêtes HTTP, effaçant l'identifiant de corrélation. D'autres ne renseignent aucune information concernant l'endpoint ou font référence à une façade générique. D'autres encore produisent un large contenu non structuré, sans utilité pour la rétro-ingénierie, mais qui ralentit le processus. Enfin, certains composants ne renseignent pas correctement un dysfonctionnement, masquant ainsi leur indisponibilité.

### 9.2.1 Limitations des efforts d'implémentation

Un système de production peut représenter un mélange hétéroclite de technologies, parfois anciennes, chacune ayant ses propres standards de développement. Pour indiquer qu'un endpoint est similaire fonctionnellement à un autre, il faut alors fournir une implémentation propre à chaque standard. Ces efforts supplémentaires de développement pour la rétro ingénierie ne seront peut-être pas acceptés.

## 9.3 Limitations de l'ingénierie du chaos

L'état de l'art de l'ingénierie du chaos présente différentes limitations de cette discipline. La suite de ce chapitre revisite les plus pertinentes pour l'approche par rétro-ingénierie.

### 9.3.1 Gestion des risques

L'approche par rétro-ingénierie permet de faciliter la découverte d'une faiblesse potentielle du système. Cette faille est peut-être connue, ou il s'agit d'un faux positif connu. Si ce n'est pas le cas, en l'absence d'une connaissance approfondie du système, il faudrait injecter une faute pour infirmer ou confirmer la faiblesse.
Pour convaincre du bienfondé de son approche, l'ingénierie du chaos propose d'injecter une faute pour démontrer que son intuition à propos de la robustesse se vérifie. Cependant, l'approche par rétro-ingénierie demande de provoquer une panne pour confirmer son diagnostic.
Il est très probable que le gestionnaire de risques demande une investigation approfondie du manque de robustesse potentiel, plutôt que d'autoriser une injection de fautes.

### 9.3.2 Personnel hautement qualifié

À première vue, la méthode permet de s'affranchir d'un personnel hautement qualifié. Lors de la découverte d'un manque de robustesse potentiel, il faut choisir entre risquer de provoquer une panne et investiguer plus avant. Sans personnel qualifié cela peut s'avérer difficile, d'autant que, se reposant sur le système de visualisation, l'organisation aura probablement affecté les ingénieurs à d'autres tâches, privant ceux-ci d'une expérience à propos de la robustesse du système.

Ce chapitre a effectué une critique de l'approche par rétro-ingénierie et a présenté ses limites. Le chapitre suivant présente les améliorations que l'on peut apporter à cette approche.

# Chapitre 10

# Améliorations

## 10.1 Augmenter et diversifier les informations

La méthode de rétro-ingénierie choisie se base sur l'extraction de données concernant l'utilisation d'endpoints dans les logs de production. D'autres sources d'informations pourraient enrichir la rétro-ingénierie.

#### Systèmes de persistance

De nombreuses applications utilisent des systèmes de persistance, par exemple des bases de données, pour enregistrer le contexte. L'utilisation de telles applications revient à consulter ou modifier l'information contenue dans les systèmes de persistance. La connaissance autour de l'utilisation de tels systèmes pourrait enrichir la rétro-ingénierie. Par exemple, l'extraction d'une même information par deux endpoints pourrait confirmer une similitude.

#### Analyse du contenu des messages

La méthode sélectionnée de traitement des logs ignore les paramètres et le contenu des requêtes. À l'instar de l'information des systèmes de persistance, celle contenue dans les requêtes pourrait infirmer ou confirmer les similitudes trouvées.

#### Logs réseau, système, etc.

Dans la recherche de points de fragilité d'un système, l'enrichissement de la rétro-ingénierie par des données systèmes et réseaux telles que les temps de réponse, l'occupation CPU ou des temps de latence serait d'une aide précieuse.

**Code source**

Certains éléments d'architecture utilisent des mécanismes pour organiser la liaison entre des endpoints. Un ESB, un Gateway définissent leurs *routes* en utilisant un fichier de description. Il suffit alors de récolter et de lire ces fichiers pour obtenir une cartographie représentant tous les endpoints attachés à ces éléments.

## 10.2 Outils de visualisation

Les systèmes largement distribués sont complexes et difficiles à appréhender. Leur étude demande des niveaux d'abstraction et de détail très variables. De plus, la quantité de données apportées par la rétro-ingénierie peut être extrêmement importante.
La méthode gagnerait en efficacité avec de nouveaux outils de visualisation capable notamment de : réorganiser les éléments selon le point de vue, le niveau de détails, les entrées de l'utilisateur ; utiliser les données de temporalité pour rejouer et mettre en évidence les variations ou changements ;

Ce chapitre a présenté quelques améliorations possibles à l'approche par rétro-ingénierie. Le chapitre suivant aborde des perspectives nouvelles pour celle-ci, notamment son utilisation en dehors du contexte de l'ingénierie du chaos.

# Chapitre 11

# Perspectives

## 11.1 Intelligence artificielle

Lorsqu'un ingénieur doit désigner une cible pour une injection de fautes, il se base sur ses connaissances du système, techniques et métiers, ainsi que sur son expérience des comportements passés du système ou de systèmes similaires. Sa décision est aussi influencée par de nombreux biais cognitifs. Les avancées dans le domaine de l'intelligence artificielle permettent actuellement d'effectuer des classements, sans préalablement fournir des catégories à l'algorithme, et ce dans des domaines très variés : organisation de photos, analyse d'habitudes d'achats, épidémiologie, etc.
De tels systèmes d'intelligence artificielle pourraient analyser les applications largement distribuées et indiquer les points de faiblesse. Ils pourraient sans doute également intervenir pour renforcer une application fragile, avant même la survenue d'une panne.

## 11.2 Cartographie des systèmes d'information

La réalisation d'une cartographie des systèmes d'information et sa mise à jour constante représentent un travail considérable. Les nouveaux processus de développement agiles tendent vers la livraison plus fréquente de mises à jour des produits logiciels.
La cartographie des systèmes d'information pourrait être plus précise et plus pérenne, si elle était réalisée, en partie du moins, par une représentation issue des logs de l'utilisation des diverses applications.

## 11.3 Utilisation des systèmes

La rétro-ingénierie réalisée à partir de logs d'utilisation permet d'exploiter, notamment les données temporelles. À partir de celles-ci, il est possible

de construire une visualisation permettant une analyse des usages des systèmes, à l'instar des analyses de trafic routier et autres systèmes de flux.

## 11.4 Optimisation de processus

De nombreux outils de workflow proposent des aides à l'optimisation de processus. Celles-ci se basent essentiellement sur des analyses statistiques de l'utilisation d'un processus. Grâce à la visualisation de l'utilisation du système, il serait possible d'offrir l'aides à l'optimisation de processus, non pas sur un seul processus, mais sur l'ensemble du système d'information.

# Chapitre 12

# Conclusion

Dans le processus de l'ingénierie du chaos, le choix de la cible d'une injection de fautes est réalisé intuitivement par un ingénieur qui doit avoir une connaissance approfondie du système. La nécessité d'employer du personnel extrêmement qualifié rend le processus peu scalable.
Ce mémoire avait comme objectif de répondre aux questions suivantes :

— En utilisant une méthode de rétro-ingénierie, est-il possible de réduire voire supprimer, l'approche intuitive lors de la sélection judicieuse d'une cible de l'injection de fautes ?
— Cette méthode offre-t-elle une aide à la décision qui permet de réduire les connaissances nécessaires à cette sélection ?

La méthode de rétro-ingénierie choisie consiste (1) à récolter les logs concernant les requêtes, (2) à effectuer des inférences bayésiennes pour deviner le système et ses faiblesses éventuelles, et (3) à produire une visualisation du système découvert.

S'il existe déjà des outils de monitoring système et réseau, ceux-ci affichent le système brut. L'apport majeur de la méthode de rétro-ingénierie proposée est de détecter les relations entre les composants permettant ainsi de révéler les fonctionnalités du système. L'approche consiste à calculer des probabilités à partir des observations des logs, en s'appuyant sur le théorème de Bayes.
En définissant le manque de robustesse par un nombre insuffisant de redondances, des composants ou des relations, la rétro-ingénierie permet également de déceler des points de fragilité.

L'approche par rétro-ingénierie peut être appliquée à de nombreux systèmes, pourvu que ceux-ci produisent des logs ou qu'ils puissent être instrumentalisés. La compréhension qu'apporte la représentation visuelle du

système étend son utilisation en dehors du cadre de l'ingénierie du chaos, notamment dans l'établissement de cartographies de systèmes d'information et dans l'étude des utilisations des systèmes d'information.

L'ingénierie du chaos par une approche par rétro-ingénierie possède ses propres limites. Pour commencer, le système doit produire des logs exploitables dans l'environnement de production. Ensuite, seuls les éléments utilisés apparaissent dans les logs, rendant invisibles les composants en sommeil. Mais la limitation principale vient de la définition même de la robustesse. Pour un système comprenant une redondance suffisante, il n'y a aucune garantie que ce système ne contienne pas de points de fragilité. Un composant peut être à la limite de ses possibilités et ne pas être en capacité d'absorber une augmentation de latence d'autres composants qui lui sont liés.

En résumé, la rétro-ingénierie apporte une représentation du système utilisé et permet de mettre en valeur des métriques à propos de celui-ci. Elle réduit les connaissances nécessaires, tant techniques que métiers, pour juger de la robustesse du système. En outre, elle fournit une image de ce qui est et non de ce qui devrait.
La méthode de rétro-ingénierie offre une visualisation sur le comportement du système utilisé et une aide à la décision, lors de la sélection d'une injection de fautes.

La rétro-ingénierie pourrait améliorer ses calculs de prédictions en intégrant de nombreuses données tels que les consommations mémoires et CPU, les temps de latences, les capacités de chaque composant, etc. Elle pourrait également dégager des relations de cause à effet entre celles-ci.
Vu le volume de données et le très grand nombre de relations possibles entre celles-ci, l'utilisation de l'intelligence artificielle, semble prometteuse.
L'utilisation de l'intelligence artificielle pourrait être étendue à l'amélioration même du système, en corrigeant de manière proactive les points de faiblesse découvert.

# Annexe A

# Modèle de maturité de l'ingénierie du chaos

**Sophistication**

1. Élémentaire
   Les expérimentations ne s'exécutent pas en production.
   Le processus est administré manuellement.
   Les résultats sont des métriques systèmes et non métiers.
   De simples événements sont utilisés.

2. Simple
   Les expérimentations s'exécutent sur un environnement de pseudo production.
   L'exécution de l'expérimentation est automatique, tandis que la surveillance et l'arrêt restent manuels.
   Les résultats sont des métriques métiers.
   De plus larges événements, tels que la latence réseau, sont appliqués sur un groupe d'expérimentation.
   Les résultats sont manuellement conservés et agrégés.
   Les expérimentations sont définies de manière statique.
   Un outillage est utilisé pour le contrôle et la comparaison de l'historique des expérimentations.
3. Sophistiquée
   Les expérimentions s'exécutent dans l'environnement de production.
   La préparation, l'exécution, l'analyse des résultats et l'arrêt manuel sont automatisés.
   Le *framework* d'expérimentation est intégré à la livraison continue (CD).
   Les métriques métiers sont comparées entre l'expérimentation et un groupe de contrôle.

La combinaison d'erreurs est appliquée sur un groupe expérimental.
Les résultats sont tracés sur la durée.
Un outillage est utilisé pour la comparaison interactive de l'expérimentation et d'un groupe de contrôle.

4. Avancée
   Les expérimentations s'exécutent à chaque étape de développement, dans chaque environnement.
   La conception, l'exécution et l'arrêt prématuré d'une expérimentation sont entièrement automatisés.
   Le *framework* est intégré avec du test A/B Dixon et al. (2013) et d'autres systèmes d'expérimentation pour minimiser le bruit.
   Les événements incluent des éléments tels que le changement d'utilisation ou la mutation d'état.
   Les expérimentations ont une portée et un impact dynamique ce qui permet de trouver les points d'inflexion.
   Les pertes de revenu peuvent être prédits à partir du résultat des expérimentations.
   Les prévisions des capacités peuvent être réalisées à partir de l'analyse des expérimentations.
   La criticité des services peut être différentiée selon les résultats des expérimentations.

**Adoption**

1. Dans l'ombre
   L'initiative n'est pas approuvée.
   Peu de systèmes sont concernés.
   Il y a peu ou pas de sensibilisation de l'organisation.
   Les expérimentations sont exécutées de manière peu fréquente.

2. Investissement
   Les expérimentations sont officiellement approuvées.
   Des ressources à mi-temps sont dédicacées à la pratique.
   Plusieurs équipes sont concernées.
   Des expérimentations sont exécutées de manière peu fréquente sur des services critiques.

3. Adoption
   Une équipe est dédiée à la pratique de l'ingénierie du chaos.
   Le résultat d'incidents est intégré au *framework* afin de créer des expérimentations de régression.

Des expérimentations sont menées régulièrement sur la plupart des systèmes critiques.
Occasionnellement, des expérimentations sont exécutées pour répondre à des vérifications suite à des incidents.

4. Attente culturelle
   Tous les services critiques sont couverts fréquemment par des expérimentations.
   La plupart des services non critiques utilisent les expérimentations.
   L'expérimentation fait partie du processus d'intégration des ingénieurs.
   La participation aux expérimentations est le comportement par défaut pour les composants systèmes et une justification est nécessaire pour en être exclu.

À titre d'exemple, les auteurs ont produit le modèle de maturité suivant.

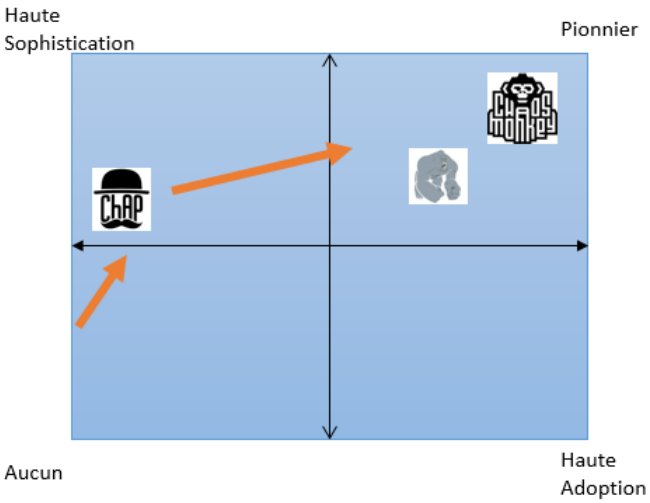


FIGURE A.1 – Modèle de maturité

# Annexe B

# *Simian Army*

Liste des composants *Simian Army* agissant au niveau du container.

1. Internationalisation
   Netflix étant disponible en différentes langues, utilisant différents jeux de caractères, ce composant vérifie la configuration et les problèmes d'exécution liés à l'internationalisation.
2. Conformité
   Ce composant à la particularité de chercher et d'arrêter les instances qui n'adhèrent pas aux bonnes pratiques. Le responsable est alors alerté de l'arrêt et une action corrective peut être mise en place.
3. Sécurité
   Ce composant est une extension du précédent. Si une instance viole les règles de sécurité ou expose une vulnérabilité, elle est immédiatement arrêtée.
4. Latence
   Induit artificiellement un délai dans la communication entre le client et le serveur RESTful pour simuler une dégradation du service. En choisissant un délai très large, ce composant peut simuler l'arrêt complet d'un service, sans intervenir sur l'instance.
5. Santé (traduit librement de *Doctor Monkey*)
   Ce composant détecte les instances qui sont en « mauvaise santé », par exemple une forte consommation du CPU. Il la signale hors service et prévient alors le responsable du composant, lui permettant de remédier aux problèmes.
6. Ressources inexploitées
   Ce composant traque les ressources inexploitées et arrête les instances inutilisées.
7. Kong
   Gardant l'analogie avec l'armée de singes, Kong fait référence au gorille géant. Ce composant est similaire a *Chaos Monkey* mais permet

de simuler un problème à l'échelle d'une zone entière d'Amazon. Il permet de vérifier que la redirection des services sur les autres zones n'a pas d'impact les clients et ne demande aucune intervention manuelle.

# Annexe C

# Code Source : Implémentation pour les cas d'étude

## C.1 chaos-re

Chaos engineering approach by Reverse-engineering.
git : https ://github.com/pcoppens/chaos-re

## C.2 Introduction

Chaos engineering leads to more resilient systems and builds confidence in a software product based on a large scale distributed architecture. The discipline is based on the definition of a steady-state and the assertion that it will remain stable when a fault is injected in production. The approach by Reverse-engineering reduce, or eliminate the intuitive approach by offer a visual representation of the system.

## C.3 Structure

- /back-end/ projet java
  - application/core/ discover Tool (use Bayes theorem)
  - application/generator/ Labo : generate a random System
  - application/input/ Read source and build an abstraction of a System
  - application/model/ Abstraction of a distributed System
  - application/output/ write a representation of the System
    - dot/ write a System to (Graphiz) dot file
    - vizceral/ write a System to Vizceral file

— /dot dot files resuult
— /vizceral Vizceral front-end

## C.4 Discover a system

1. Get log files or build a generated system (/back-end/generator/DistribuedSystem.main)
2. Run a process (application/ProcessForemLogs or application/ProcessLogs)
3. See representation (generate images from dot/files.dot or run Vizceral : See vizceral/README.md )

# Bibliographie

(2017), *Using Firewall and Application Logs*, chapter 9, 211–243. John Wiley and Sons, Ltd, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119329190.ch9.

Alvaro, P., J. Rosen, K. Ram, and L. Oldenburg (2015), "Molly : Lineage-driven fault injection." URL https://github.com/palvaro/molly.

Alvaro, Peter, Kolton Andrus, Chris Sanden, Casey Rosenthal, Ali Basiri, and Lorin Hochstein (2016), "Automating failure testing research at internet scale." In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, 17–28, ACM, New York, NY, USA, URL http://doi.acm.org/10.1145/2987550.2987555.

Alvaro, Peter and Severine Tymon (2017), "Abstracting the geniuses away from failure testing." 61, 54–61.

artillery.io (2015), "https ://artillery.io/chaos-lambda/."

Bachelet, Rémi (2015), "Rémi bachelet maître de conférences." URL http://rb.ec-lille.fr/.

Barbosa, Raul, Johan Karlsson, Henrique Madeira, and Marco Vieira (2012), *Fault Injection*, 263–281.

Basiri, A., N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal (2016), "Chaos engineering." *IEEE Software*, 33, 35–41, URL doi.ieeecomputersociety.org/10.1109/MS.2016.60.

Blohowiak, Aaron, Ali Basiri, Lorin Hochstein, and Casey Rosenthal (2016), "A platform for automating chaos experiments." URL https://medium.com/netflix-techblog/chap-chaos-automation-platform-53e6d528371f.

Brooks, Frederick, Jr (1987), "No silver bullet essence and accidents of software engineering." 20, 10–19.

Ciancutti, John (2010), "5 lessons weve learned using aws." URL `https://medium.com/netflix-techblog/5-lessons-weve-learned-using-aws-1f2a28588e4c`.

Cleve, Anthony, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens H. Weber (2015), "Understanding database schema evolution : A case study." *Sci. Comput. Program.*, 97, 113–121.

Dixon, E., E. Enos, and S. Brodmerkle (2013), "A/b testing." URL `https://www.google.com/patents/US8583766`. US Patent 8,583,766.

Ellson, John, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull (2003), "Graphviz and dynagraph – static and dynamic graph drawing tools." In *GRAPH DRAWING SOFTWARE*, 127–148, Springer-Verlag.

ESIPE (2013), "Introduction au reverse engineering." URL `http://igm.univ-mlv.fr/~dr/XPOSE2013/reverse_engineering/generality.html`.

Evans, Josh (2016), "Mastering chaos - a netflix guide to microservices." QConSF.

Foote, Brian and Joseph Yoder (2003), "Big ball of mud."

G., O'Regan (2017), *Software Testing. In : Concise Guide to Software Engineering.*, chapter 7. Software Testing, 105–106. Springer International Publishing.

Gámez-Díaz, Antonio, Pablo Fernández Montes, and Antonio Ruiz-Cortés (2015), "Towards sla-driven api gateways."

Guinard, Dominique, Iulia Ion, and Simon Mayer (2016), "In search of an internet of things service architecture : Rest or ws-* ? a developers ' perspective."

Halin, Axel, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry (2017), "Test them all, is it worth it ? a ground truth comparison of configuration sampling strategies."

Hoang, Lê Nguyên (2018), *La Formule du Savoir : Une philosophie unifiée du savoir fondée sur le théorème de Bayes (French Edition)*. EDP Sciences, URL `https://www.amazon.com/Formule-Savoir-philosophie-unifi%C3%A9e-th%C3%A9or%C3%A8me-ebook/dp/B07DRGG31Y?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B07DRGG31Y`.

Hochstein, Lorin and Communauté open source (2017), "Netflix/chaosmonkey."

Hohpe, Gregor and Bobby Woolf (2003), *Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, URL `https://www.amazon.com/Enterprise-Integration-Patterns-Designing-Deploying/dp/0321200683?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0321200683`.

Inanzzz (2017), "http ://www.inanzzz.com/index.php/post/3c6n/setting-up-elasticsearch-logstash-and-kibana-elk-stack-on-ubuntu-14-04."

Inc., Gremlin (2016), "https ://www.gremlin.com/."

JHipster (2013), "https ://www.jhipster.tech/." URL `https://www.jhipster.tech/`.

Kawaguchi, Kohsuke (2011), "https ://jenkins.io/."

Kolton Andrus, Ben Schmaus, Naresh Gopalani (2014), "Fit : Failure injection testing." URL `https://medium.com/netflix-techblog/fit-failure-injection-testing-35d8e2a9bb2`.

Kosewski, L., J. Tatelman, J. Reynolds, and C. Rosenthal (2015), "Flux : A new approach to system intuition." URL `https://medium.com/netflix-techblog/flux-a-new-approach-to-system-intuition-cf428b7316ec`.

Lew, K. and S. Narayanan (2017), "Lessons from building observability tools at netflix."

Lorin Hochstein, Casey Rosenthal (2016), "Netflix chaos monkey upgraded." URL `https://medium.com/netflix-techblog/netflix-chaos-monkey-upgraded-1d679429be5d`.

Macero, Moises (2017), *Learn Microservices with Spring Boot : A Practical Approach to RESTful Services using RabbitMQ, Eureka, Ribbon, Zuul and Cucumber*, chapter The microservices journey through tools, 214.

Maedche, A., A. Botzenhardt, and L. Neer (2012), *Software Product Management. In :Software for People*. Springer, Berlin, Heidelberg.

Mazzara, Manuel and Bertrand Meyer (2017), *Present and Ulterior Software Engineering*. Springer International Publishing.

Microsoft (2012), "https ://www.visualstudio.com/fr/team-services/."

Microsoft (2018), "https ://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-controlled-chaos."

Montesi, Fabrizio and Janine Weber (2016), "Circuit breakers, discovery, and API gateways in microservices." *CoRR*, abs/1609.05830, URL `http://arxiv.org/abs/1609.05830`.

Nakama, Heather (2015), "Inside azure search : Chaos engineering." URL `https://azure.microsoft.com/en-us/blog/inside-azure-search-chaos-engineering/`.

Nasr, Salah and Hassen Mekki (2018), "Chaos engineering and control in mobile robotics applications." In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - Volume 2 : ICINCO,*, 364–371, INSTICC, SciTePress.

Netflix (2015), "https ://www.spinnaker.io/."

Netflix (2016), "Annual report pursuant to section 13 or 15(d) of the securities exchange act of 1934 for the fiscal year ended december 31, 2016."

Parrish, Kyle and Dave Halsey (2015), "Too big to test : Breaking a production brokerage platform without causing financial devastation." O'Reilly Velocity Conference.

Pathania, Nikhil (2017), *Pro Continuous Delivery*, chapter Elements of Continuous Delivery, 1–21. O'Reilly.

Pierre, Dagnelie (2012), *PRINCIPES D'EXPERIMENTATION Planification des expériences et analyse de leurs résultats*, chapter Les facteurs et les traitements ou objets, 36. LES PRESSES AGRONOMIQUES DE GEMBLOUX, A.S.B.L.

Rahman, Akond, Amritanshu Agrawal, Rahul Krishna, Alexander Sobran, and Tim Menzies (2017), "Continuous integration : The silver bullet ?"

R.Anusooya1, J.Rajan, and S.A.V.SatyaMurty (2015), "Importance of centralized log server and log analyzer software for an organization." *International Research Journal of Engineering and Technology (IRJET)*, 2.

Reynolds, J., T. Cichocinski, et al. (2017), "https ://github.com/netflix/vizceral."

Rosenthal, Casey, Lorin Hochstein, Aaron Blohowiak, Nora Jones, and Ali Basari (2017), *Chaos Engineering, Building Confidence in System Behavior through Experiments*. O'Reilly.

Russell, Craig (2017), "https ://medium.com/@craig552uk/how-to-choose-an-api-gateway-ac64f04f3683."

Sendmail (2007), "E-mail auditing, logging and reporting, white paper."

SridharJason, Nigamanth and O. Hallstrom (2006), "A behavioral model for software containers." In *Lecture Notes in Computer Science Conference : Fundamental Approaches to Software Engineering*, volume Vol. 3922, 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software.

Strauss, Daryll (1998), "Linux helps bring titanic to life." URL `https://www.linuxjournal.com/article/2494`.

Tucker, H., L. Hochstein, N. Jones, A. Basiri, and C. Rosenthal (2018), "The business case for chaos engineering." *IEEE Cloud Computing*, 5, 45–54, URL `doi.ieeecomputersociety.org/10.1109/MCC.2018.032591616`.

UniKnow (2016), "https ://uniknow.github.io/agiledev/site/0.1.10-snapshot/eip/correlation-id."

Voas, Jeffrey (1998), "Fault injection for the masses." 30, 129 – 130.

Warner, Richard and Robert Sloan (2011), "Vulnerable software : Product-risk norms and the problem of unauthorized access."

Wilms, Benjamin (2018), "https ://codecentric.github.io/chaos-monkey-spring-boot/2.0.0/."

Yuan, Ding, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm (2014), "Simple testing can prevent most critical failures : An analysis of production failures in distributed data-intensive systems." In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, 249–265, USENIX Association, Berkeley, CA, USA, URL `http://dl.acm.org/citation.cfm?id=2685048.2685068`.

Yury Izrailevsky, Ariel Tseitlin (2011), "The netflix simian army." URL `https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116`.